12-2017

Image Down-Scaler Using the Box Filter Algorithm

Vaishnavi Parthipan
vp4451@rit.edu

Image Down-Scaler Using The Box Filter Algorithm

by
Vaishnavi Parthipan

Graduate Paper

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in Electrical Engineering

Approved by:

Mr. Mark A. Indovina, Lecturer
Graduate Research Advisor, Department of Electrical and Microelectronic Engineering

---

Dr. Sohail A. Dianat, Professor
Department Head, Department of Electrical and Microelectronic Engineering

Department of Electrical and Microelectronic Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
December 2017

I would like to dedicate my project work to my family members and friends who were constant support through out my education in Rochester Institute of Technology.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this paper are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This paper is the result of my own work and includes nothing which is the outcome of work done by others, except where specifically indicated in the text.

Vaishnavi Parthipan

December 2017

Acknowledgements

I would like to thank my advisor, professor, Mark A. Indovina, for all of his constant support, guidance and feedback throughout the entirety of this project.

Abstract

One of the indispensable aspects of digital image processing is the requirement of varied image resolutions. To achieve varied resolution, scaling comes into picture. Two important applications of scaling is good pictorial quality for human interpretation and processing of digital images for storage, transmission and for representation of autonomous machine perception. This paper focuses on the transmission application. The size of the image if reduced occupies less space in the communication medium thus reducing the bandwidth requirement. And also the server space and the processing power of the image is reduced greatly. The standard for digital television transmission over terrestrial, cable and satellite networks is defined by Advanced Television Systems Committee (ATSC), with either 704 $\times$ 480 or 640 $\times$ 480 pixel resolutions, at 24, 30, or 60 progressive frames per second. This paper proposes a monochrome and colored image down scaling core with memory banks for accessing the image pixels. A 704 x 480 pixel resolution image was used. The core has minimum complexity and was developed in Hardware Descriptive Language (HDL). Its been benchmarked in various ASIC technologies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digitizing an image has many benefits in transmitting, processing, and storing the image data [1]. Digital images are widely used in the field of consumer electronics and medical industry. Digital television (DTV) is an advanced broadcasting technology for television where digital image transmission and processing plays a vital role. Unlike earlier television technology where video and audio were analog signals, DTV in contrast is used for television signal transmission with sound channel and by digital encoding. After the evolution of color television in 1950s, DTV marks one of the significant innovations. Digital television provides broadcasters to offer television with better picture and sound quality along with multiple channels of programming.

Different countries have different standards for broadcasting digital television. Advanced Television System Committee (ATSC) uses eight-level Vestigial Side-Band (8VSB) for terrestrial broadcasting. This standard has been adopted by six countries: United States, Canada, Mexico, South Korea, Dominican Republic and Honduras. Multiple channel transmission takes place in a single bandwidth. Due to the requirement of multi-channel video broadcast, the pressure is on the bandwidth to carry all the channels. But we know

the bandwidth for video communication is minimal and always in need for more. The never ending growth in the area of digital communication fueled many communication algorithms to meet the need of bandwidth. One such communication algorithm is scaling of images, which helps to reduce the size before transmission.

ATSC uses 640 X 480-pixel or 702 X 480-pixel resolution for transmission over terrestrial and satellite networks. Digital terrestrial broadcasting can be designed to work with roof-top antennas but also with small antennas built into portable devices and for mobile reception [2]. To reduce the bit rate while transmitting, the resolution of the images in the video frames are reduced. At receiving end the image is interpolated by pixels to reconstruct image to the size of destination display monitor. This paper concentrates on the challenging and significant facet i.e., down scaling of image to address the bandwidth need. The basic concept of image scaling is to re-sample a two-dimensional function on a new sampling grid. There are various down-scaling algorithm available, out of which this paper implements box filter for various reasons. Box filter algorithm is not for the images perceived for visual content. It is mainly for the transmission purpose.

The box filter is a simple filter that can be efficiently calculated by storing partial sums [3]. Another reason for choosing box-filter is the computational complexity. The computational complexity is very low compared to other filters such as bi-cubical and bi-linear filters used for down-scaling. If implemented in an Application Specific Integrated Circuit (ASIC), or a Field Programmable Gate Array (FPGA), the number of logic gates required will be very less, but in contrast it needs more on chip memory register banks for storage of the partial sum. The use of on-chip memories nowadays are expected to increase continuously based on the future generation high performance and portable devices [4]. An FPGA is an integrated circuit, which is configured after manufacturing using a Hardware Descriptive language (HDL). In this paper box filter algorithm is being used with on chip

memory for storage of original pixels and the intermediate partial sum pixels. The down scaling core is coded in HDL (Hardware Descriptive Language) and has been benchmarked in a ASIC core module targeting 32 nm, 65nm, 90nm and 180nm technologies.

1.1 Organization

The organization of this paper is as follows: Chapter 2 discusses about the comparison between different algorithm used for down-scaling, Chapter 3 gives an insight into the Technical Concepts related to digital image down-scaling and Chapter 4 explains the Algorithm used in this paper. Next, Chapter 5 explains the Hardware Implementation of the algorithm, Chapter 6 discusses about the Design Flow and Test-bench and the final Chapter 7 gives the Result and Conclusion of this paper.

Chapter 2

Literature Review

There are three main methods used for image down-scaling: bi-linear, bi-cubical and Nearest neighbor algorithm. This section deals with these different methods and why box filter is a better choice for image down-scaling.

One of the widely used algorithms for down-scaling is the Bi-cubical algorithm. Bi-cubical interpolation is used in a two dimensional grid for interpolating pixels. This is derived from the cubic interpolation. As the name depicts, the 'Bi' in this algorithm refers to computing two 1 dimension operations: vertical and horizontal computation. This algorithm is used is for most of images, assuming the image is size not to small, or that this image is not highly detailed, or if the edges of the image are to be kept smooth. Moreover the contrast and artifacts are increased using the bi-cubic algorithm. The operation of bi-cubic convolution requires the calculation of 16 weighted coefficients generated from 16 pixels of source image [5]. The coefficient, c, of its corresponding source pixel can be

obtained from the Equation 2.1.

$$c = \begin{cases} 1 - 2|d|^2 + |d|^3 & , \ 0 \leq |d| < 1 \\ 4 - 8|d| + 5|d|^2 - |d|^3 & , \ 1 \leq |d| < 2 \\ 0 & , \ 2 \leq |d| \end{cases} \tag{2.1}$$

Then four virtual pixel(V_0, V_1, V_2 and V_3) are calculated by either vertical or horizontal interpolation as shown in Equation 2.2.

$$\sum_{i=0}^{3} \quad A_{i,j} \quad \text{X} \quad vc_i \tag{2.2}$$

where $vc_i$is the vertical co-efficient of one of the (4 X 4) pixels $A_{i,j}$.

As described in [5], the operation of this filter in addition to the calculation of the weighted co-efficient, also requires the co-ordinates of the pixel. The number of adders and multipliers used for implementing the co-efficient calculation are high. To achieve bi-cubical interpolation with good results, a high number of resources are required. There is additional hardware to satisfy the need of the complexity such as the Pixel co-ordinate generator, horizontal and vertical pixel generator, and a virtual pixel buffer. The vertical pixel generator stores the calculated intermediate pixel in the virtual pixel buffer. Since most of the image doesn't come with inbuilt memory, the number of memory reads and writes are high. Memory reads are costly in certain cases. This is avoided in the proposed box filter with register banks to store pixel values. If a scale factor of 2×2 is used with bicubic interpolation, the hardware implementation of an image expansion method can be significantly simplified [6].

Bi-cubic interpolation is a robust technique compared with other techniques such as the BOX FILTER [3] and WINSCALE algorithm [7], these other techniques are simpler and require

less resources and are more feasible for implementation in sequential processors [8]. Due to less complexity BOX FILTER is faster compared to bi-cubic filter. Note that the bi-cubic interpolation method is somewhat complicated as compared to bi-linear interpolation [9].

In contrast to the bi-cubic algorithm which takes 16 pixels (4X4) , the bi-linear algorithm uses only 4 pixels (2 X 2) in account. Bi-linear is extension of linear interpolation. The main motive is to perform linear interpolation in one direction and then repeat the same in another direction. This method reduces the contrast of the image, and never produces a satisfying image, combining a fuzzy overall look with jagged edges and motion artifacts [10]. This algorithm takes the weighted average of the four closest pixels, and maps it to the specified output co-ordinates. The speed at which the output pixels are calculated is low compared to other techniques [11]. Apart from having the slow calculation unit, the technique described in this paper has an extra graphical processing unit to accelerate the process. The calculation time was faster but only after giving away resources. One other disadvantage of using the bi-linear algorithm is that its more efficient for images with lesser pixels. Since it computes the weighted average of (2 X 2) pixels, the amount of time it would take to compute the entire image would be significantly high. The weight on each of the 4 pixel values is based on the computed pixel's distance (in 2D space) from each of the known points.

Let us consider four pixels $R_{1,1}$, $R_{1,2}$, $R_{2,1}$ and $R_{2,2}$ in a (x, y) co-ordinate system. Now let Q1 be weighted average of $R_{1,1}$ and $R_{2,1}$. Similarly Q2 be weighted average of $R_{2,1}$ and $R_{2,2}$. First, the $Q1$and $Q2$ values are calculated using the Equation 2.3 and 2.4. Here $x1$, $x2$, $y1$ and $y2$ represent the position in the co-ordinate system.

$$Q1 = ((x2–x)/(x2–x1)) * R_{1,1} + ((x–x1)/(x2–x1)) * R_{2,1} \tag{2.3}$$

$$Q2 = ((x2–x)/(x2–x1)) * R_{1,2} + ((x–x1)/(x2–x1)) * R_{2,2} \tag{2.4}$$

Second, after the calculation of Q1 and Q2, the down-scaled pixel value P is calculated by the Equation 2.5.

$$P = ((y2–y)/(y2–y1)) * Q1 + ((y–y1)/(y2–y1)) * Q2 \tag{2.5}$$

So if we take a closer look at Equation 2.3 and 2.4, we see they deal with the x-axis of the co-ordinate system. Thus, the interpolation happens in one direction and then the interpolation is repeated in the other direction y, as given in Equation 2.5. The paper [12] uses the bi-linear algorithm and requires eight multiplications, three additions and four subtractions.

One weakness of bi-linear, bi-cubic and related algorithms is that they sample a specific number of pixels. When down scaling below a certain threshold, such as more than twice for all bi-sampling algorithms, the algorithms will sample non-adjacent pixels, which results in losing data and causes rough results. And these traditional downscaling algorithms mainly address the aliasing problem [13].The trivial solution to this issue is box sampling, which is to consider the target pixel a box on the original image, and sample all pixels inside the box. This ensures that all input pixels contribute to the output. The major weakness of this algorithm is that it is hard to optimize. But these two algorithms are useful when high quality image is required. But one of the main drawback of such algorithm is the amount of resources used. Where as other techniques are simpler and need less resources. The common thing among all the three algorithms is after certain input latency, the pixel value is calculated at the data rate. Area averaging or box filter on the other hand is simple, fast and near optimal.

Chapter 3

Technical Overview

This chapter gives an elementary introduction to bit-mapping, aspect ratio and resolution aspect of an image.

3.1 Bitmaps

Bitmaps are defined as a rectangular mesh of cells called pixels. Each pixel in an image carries the color of the designated co-ordinate in a image [14]. The value of the pixel in each cell gives the color value. Bit maps are defined by only two characteristics namely the number of pixels and the information content of the pixel (color depth). Apart from the number of pixel and pixel content (two parameters), there are other parameters which are derived from these two fundamental parameters.

The layout of the bitmap is generally aligned horizontally and vertically. In most cases the bitmap is used to represent an image in computer program.

3.2 Color depth

Each pixel contains information about the color of the image. This information content is called color depth. Color depth refers to the number of bits used to represent color information of either a single pixel or the number of bits of each color component of a single pixel depending on the image format. Images with higher bit depths can encode more shades or colors since there are more combinations of 0's and 1's available [15].

3.2.1 1-bit (black and white)

This is the smallest possible content that can be held by a pixel [14]. The bit 0 is represented for white and 1 describes the black in that pixel. The resulting bitmap is called monochrome or black and white image. Since there are only two values, the pixel can be used to represent 0 as black and 1 as white also.

3.2.2 8-bit grey

In this case each pixel takes 8 bit or 1 byte of data to store the color value. Then the values of the pixels will be from 0 to 255. If these different values are mapped on to a ramp of 1 to 8 bit, then these are called as greyscale images shown in Fig 3.1. 0 is normally black and 255 white. The in-between values are the grey levels, for example, in a linear scale 127 would be a 50% grey level.

In any particular application the range of grey values can be anything, and it is most common to map the levels 0-255 onto a 0-1 scale; some programs will map to a 0-65535 scale. Digital images requires, at a minimum an 8-bit per pixel where 256 discrete levels or steps of tones describe the entire tonal range from black to white [16].

Figure 3.1: 8-bit gradient

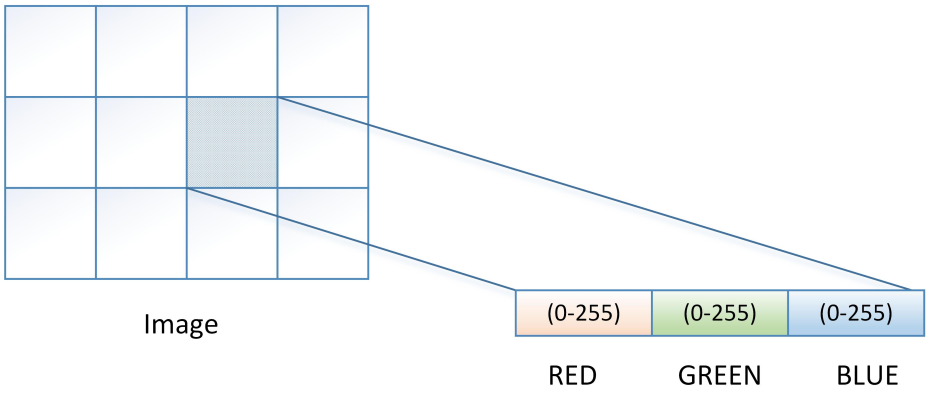

Figure 3.2: RGB color Format

3.2.3 24-bit RGB

The 24-bit RGB format is used to represent a colored image. Here 8 bits are allocated to each red, green, and blue component of a pixel. In each component the value of 0 refers to no contribution of that color, 255 refers to fully saturated contribution of that color. Since each component has 256 different states there are a total of 16777216 possible colors as shown in Fig. 3.2.

In this paper the box filter algorithm is used for two type of images, greyscale and

colored image. The greyscale image is represented in a 8-bit format and the colored image is represented in 24-bit format.

3.3 Resolution

Pixels themselves don't have a dimension, so the resolution attribute of bitmap comes in the picture for visually viewing or printing. Resolution is normally specified in pixels per inch but could be in terms of any other unit of measure. Most printing processes retain the Pixels Per Inch (PPI) or Dots Per Inch (DPI) units for historical reasons. In digital printing world, DPI is defines the amount of pixels that can be accommodated in line or area within a span of 1 inch. PPI has similar definition to DPI, but here the amount of pixel referred is in a digital image. The resolution may be specified as two numbers, the horizontal and vertical resolution.

The information in the bitmap content is different from the perspective of resolution of that image, given a constant color depth then the information content between different bitmaps is only related to the number of pixels vertically and horizontally. The quality of the image, when printed or displayed as digital image hugely depends on the resolution. In-order to modify the overall image size the resolution of the image is changed.

As an example consider one bitmap which is 200 pixels horizontally and 100 pixels vertically. If this bitmap was printed at a resolution of 100 DPI, then it would measure 2 inches by 1 inch. If however the same bitmap was printed at 200 DPI then it would only measure 1 inch by 1/2 an inch.

3.4 Aspect ratio

The ratio of width to the height of an image is called aspect ratio. It is usually represented by two numbers with a colon separation, like 4:3. If there are three different images say, 4 inches wide and 3 inches high, 4 mm wide and 3 mm high. All have the same aspect ratio. This means the size of the image does not concern the aspect ratio. The international standard aspect ratio for digital television is 16:9. This aspect ratio one of the wide screen ratio that is supported by the [1]. Pixels are usually considered as a square, all though they have other aspect ratios.

The image used in this paper is (704×480) pixels, which is 704 pixel horizontally and 480 pixels vertically. Downsizing image and video content of resolutions 704 X 480, 640×480 and 320×240 are of interest as many mobile devices have displays with such resolutions [17]. These resolutions translate to downsizing factors of $4/9 \times 3/8$, $4/9 \times 1/3$ and $2/9 \times 1/6$, respectively . Thus, the aspect ratio of the image used in this paper is 4:3.

Chapter 4

Box Filter algorithm

A box filter, also known as "moving average", is a simple linear filter with a square (or rectangular) kernel and all kernel coefficients equal and it is the quickest filter algorithm, but it lacks smoothness of a Gaussian filter [18]. The number of logic gates required for this algorithm is less, however in this paper register banks that are required to store the partial sums increase the area required. These register banks are reused when compared to the naive implementation where partial sums are not stored. Every time a partial sum is necessary, it is recalculated on demand in such implementation. Higher resolution images require more register banks, since the partial sum corresponds to the size of image width.

The computational complexity of image filtering depends on the complexity and size of the filter. There are two different approaches for this filter : integral image algorithm [19] and summed area algorithm [20]. In this paper we are using the summed area algorithm. Before starting the explanation of the filter method, lets go through some terminologies. Source image is the original image before scaling up/down. Target image refers to the scaled image. The region of the target pixel in the source that are being calculated currently is called a filter window.

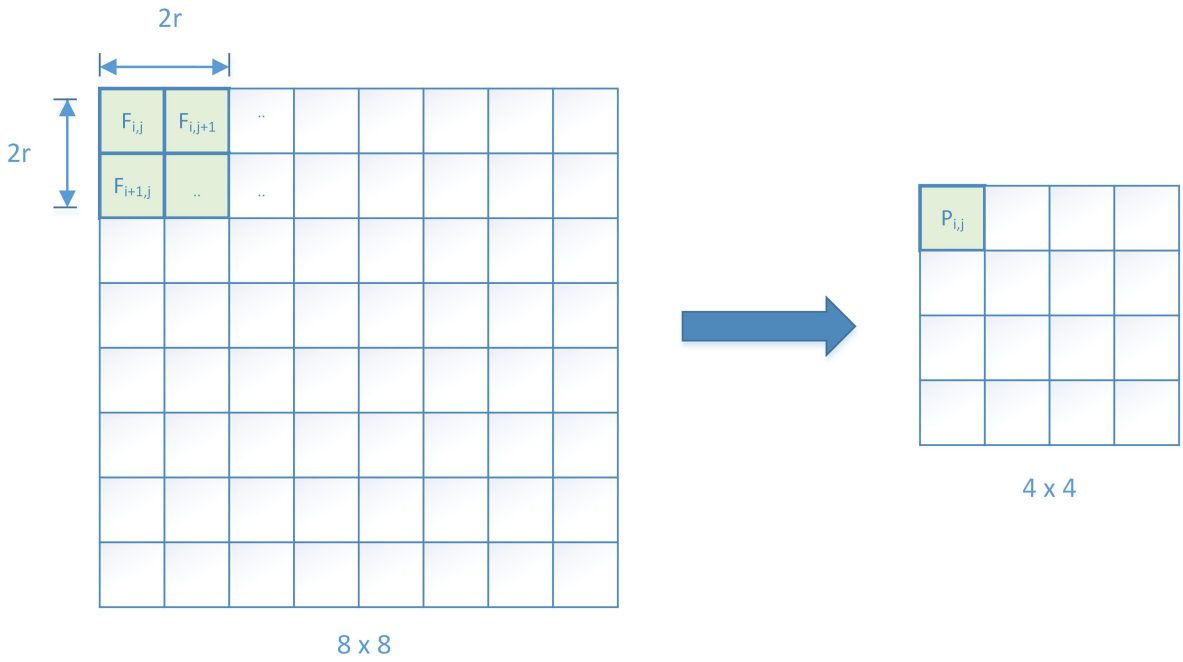

Figure 4.1: Box filter method

Box filter is a simplest linear filter. Each pixel calculated is the average of the pixel values in a square (filter window) centered at that pixel. The difference between box filter and other linear filter is that the rest of them use a weighted average. The calculation the target pixel is by moving the averaging window either horizontally or vertically. In this paper the window is moved horizontally as shown in Fig.4.1.

Let P be the pixel value of the targeted pixel. From Fig.4.1 it can be noted that the window size is 2 X 2. Here the window size is an even number of rows and columns i.e, $2r$ X $2r$, where r is the window radius. It is also possible to have odd number of rows and columns. In that case the window size will be $(2r + 1)$ X $(2r + 1)$. Let $F_{i,j}$, for $i,\ j = 1, ..., n$ denote the pixel values in the image. A box filter can be with a complexity

independent of a filter radius [21]. The formula for the targeted pixel value is given by

$$P_{i,j} = \sum_{k=0}^{m} \sum_{l=0}^{m} w_{k,l} \; F_{i+k,j+l} \tag{4.1}$$

for $i,\ j = (m+1), ...(n-m)$.

Let m = 1, then the window over which the averaging is carried out is 2 X 2 and $P_{i,j}$is given by

$$P_{i,j} = W_{0,0} \quad F_{i,j} \quad + \quad W_{0,1} \quad F_{i,j+1} \quad + \quad W_{1,0} \quad F_{i+1,j} \quad + \quad W_{1,1} \quad F_{i+1,j+1} \tag{4.2}$$

Here for the box filter the weights distributed for all the four pixels are equal. This then becomes the average of the four pixels.

$$P \quad = \frac{1}{(2r)^2} \quad F_{i,j} \quad + \quad F_{i,j+1} \quad + \quad F_{i+1,j} \quad + \quad F_{i+1,j+1} \tag{4.3}$$

The implementation of the filter in this paper has 4 register banks to store the original pixels and 2 register bank to store the partial sum of the pixels. One advantage of this implementation is there is no memory access. As implemented in this paper, the resolution is constant, thus the register banks to store the original pixel before calculation and to store the partial sum are fixed.

Chapter 5

Hardware Architecture

Detailed information about each block of the image scaler is explained here.

5.1 Register Bank Architecture

Considering the width of the image, the size of the register bank is decided. Here there are two different register banks, one for storing the source pixels, i.e., input pixels, and the second is used to store the intermediate pixels or virtual pixels that are used for further down-scaling.

Let us discuss the two different register banks used here.

5.1.0.1 REGISTER BANK - Horizontal input pixel register bank

This register bank is used to store the pixel value in horizontal manner. The image is read row by row. Each row contains 704 pixels. Therefore, there are 704 unpacked 8-bit width register blocks for greyscale images, and 24-bit wide registers for colored images. Along with the input pixel, a series of address values are given to address each block of the

Figure 5.1: Horizontal buffer

register bank. This address is carried throughout the design, for storage of virtual pixels as well as for the output pixel. The control signals wr_en and rd_en are active HIGH signals. Apart from the write and read enabling operations, these signals are also used as to synchronize operations between register banks. Fig. 5.1 shows the horizontal register bank block diagram.

These control signal are driven by a switch that decides which register bank needs to be written with input pixel values and which register bank needs to be read. More details about the Control signal is given in the section 5.3. There are total of four horizontal input pixel buffers. A pair of register banks is used for the storage of the consecutive rows at a given HIGH wr_en. Once these two registers are filled, the corresponding rd_en stroke is enabled to read those registers. The pair of registers are written and read one after the other. Total of 704 clock cycles is required to store two rows of pixels in a pair of register banks. Once the register banks are filled, a Done_cs signal is asserted to indicate that the entire register is filled with a row of pixels.

Now coming to the output part of the register bank, if the control signal toggles to rd_en, the data_out outputs the pixel value corresponding to the address_in given at the

Figure 5.2: Virtual buffer

input. Thus, there is simultaneous read and write enabled in the block.

5.1.0.2 REGISTER BANK - virtual pixel register bank

These are similar to horizontal register banks but the size of number of registers varies. The number of registers are half the number of registers used in horizontal buffers. When the down-scaler processes each pair of horizontal register banks, it filters 2 X 2 pixels producing 352 pixels. That is the reason for half the number of registers used in virtual register banks. The virtual register bank block diagram is shown in the Fig. 5.2. The address_in for the virtual register bank comes from the down-scaler. The wr_en and rd_en signals are from the control block.

If the pixels stored in the virtual register banks are collected, they will produce an image of size 352 X 240 pixels.

5.2 Filter : Pixel down-scaler

This block is used to downscale the input pixels. It takes two clock cycles to produce the average of four pixels. This implementation is similar to the WINSCALE algorithm [7] where the filter window frame moves over the pixels that are to be averaged. But here there is no concept of filter window, wherein here one pixel from each of the horizontal register bank pair is read and added every cycle. Then it reads the next set of pixels and

Figure 5.3: Filter: Pixel downscaler

adds. Once addition of four pixels are done, the filter then shifts the total sum to produce a partial sum. The output produced is every other clock cycle. Meaning it takes two clock cycle to produce the output. This filter apart from averaging also generates the output address to store these pixels. This address is send to the virtual register bank for storage of intermediate pixel. Pixel down-scaler block diagram is shown in the Fig. 5.3.

The input to the filter is from two horizontal register bank, so there are two filter for the four horizontal register banks. And there is one filter for the two virtual register banks. The net result is to down-scale sixteen pixels to have a resolution of 176 X 120 pixels is achieved by a total of three filters.

The enable signal is triggered from the test-bench, when high the two pixel that are read will be added. The addition of the two pixels is synthesized as ripple carry adder. These ripple carry adders are made up of half adders and full adders. The Fig. 5.4 shows the ripple carry adder used to add the two incoming pixels.

5.3 Control Switch

Fig. 5.5 shows the control switch block. The control switch produces the wr_en and rd_en signals using two multiplexers that select which write or read signal needs to enabled, such that a pair of registers starts to process data pixels. This is the only block that controls

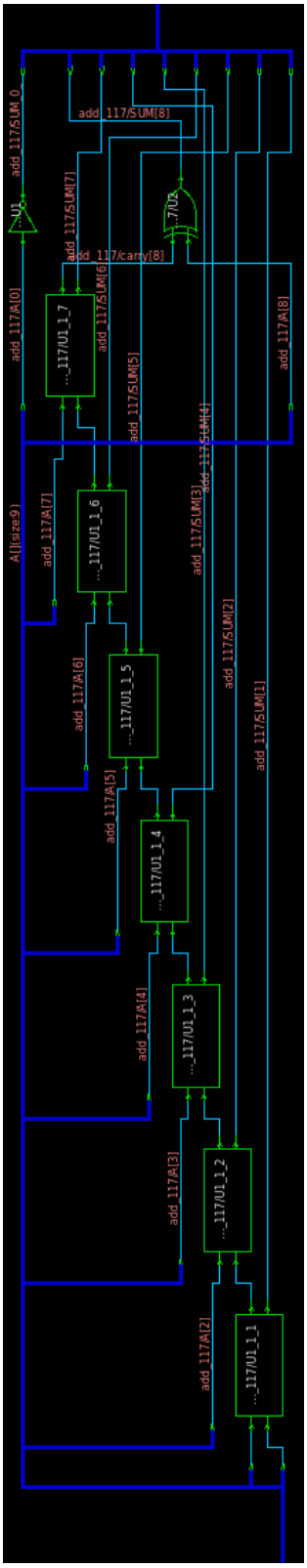

Figure 5.4: Ripple carry adder for two pixel

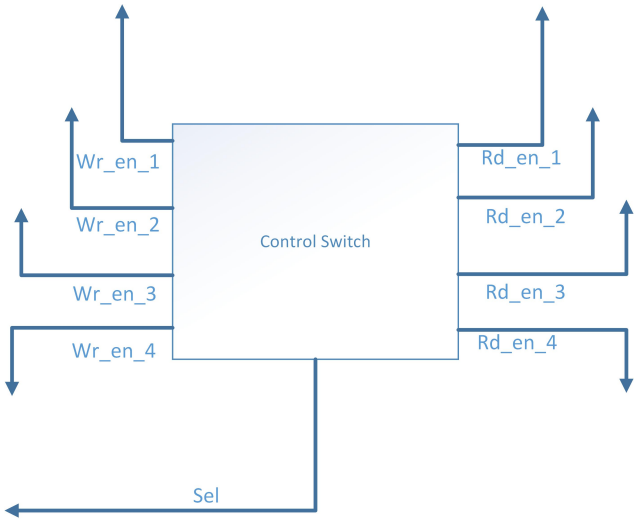

Figure 5.5: Control switch

all the write and read signals across the Image Down-Scaler module. The select signal, sel, enables which read and write signals need to be executed. The select signal comes from the test-bench. There are total of four wr_en signal lines, one for each horizontal register block. This same wr_en signal is given to the virtual register bank, but wr_en are active low for virtual register banks. In that way once the horizontal register banks fills with the wr_en at active high, then when the wr_en goes low it triggers the virtual register bank to write the intermediate pixel values. Similarly there are four rd_en signals that are used by four horizontal registers and virtual registers.

Chapter 6

Design and Test-bench

This section presents the top level design flow and the test-bench used for design verification.

6.1 Top-Level Design flow

The top level design of the Image Down-Scaler module is shown in the Fig. 6.1.

As shown in the design two rows of pixel starting from the first row is given to the first and second horizontal register bank. Once they are filled, the next set of registers (3 & 4) are fed with rows from the test-bench, and the rd_en signal is generated for the 1^{st} & 2^{nd} registers by the control signal block. The Fig. 6.2 shows the flow of data throughout the design. When the second set of registers are being written, the filter starts reading the first set of register values and starts computing the output, which is average of four pixel. In Fig. 6.2 the writing into register is shown in green and the filter operation done before virtual register is shown in red color. And when again the first set of registers are being fed, the 3^{rd} & 4^{th} register values are filtered out.

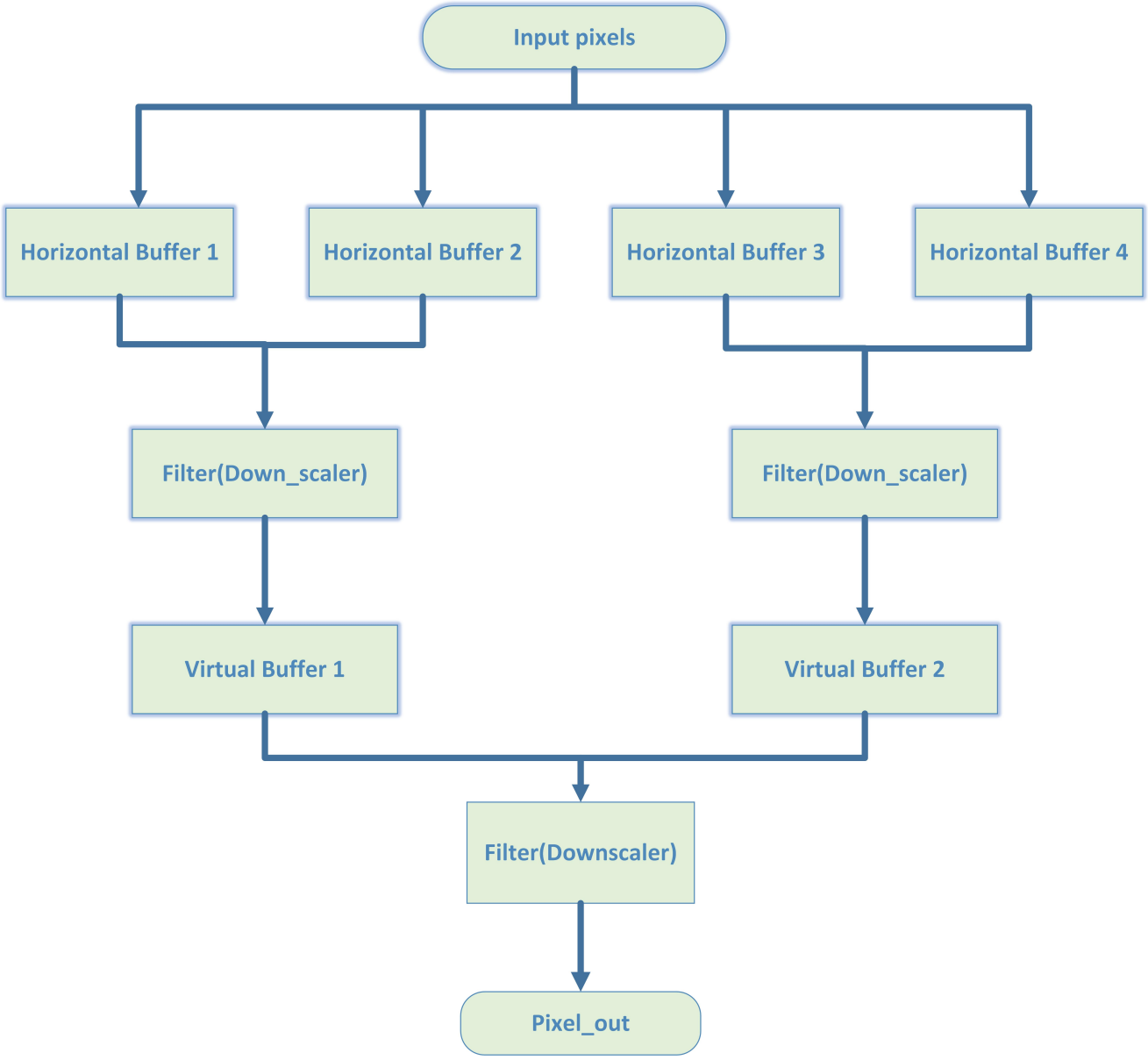

Figure 6.1: Top-level Design

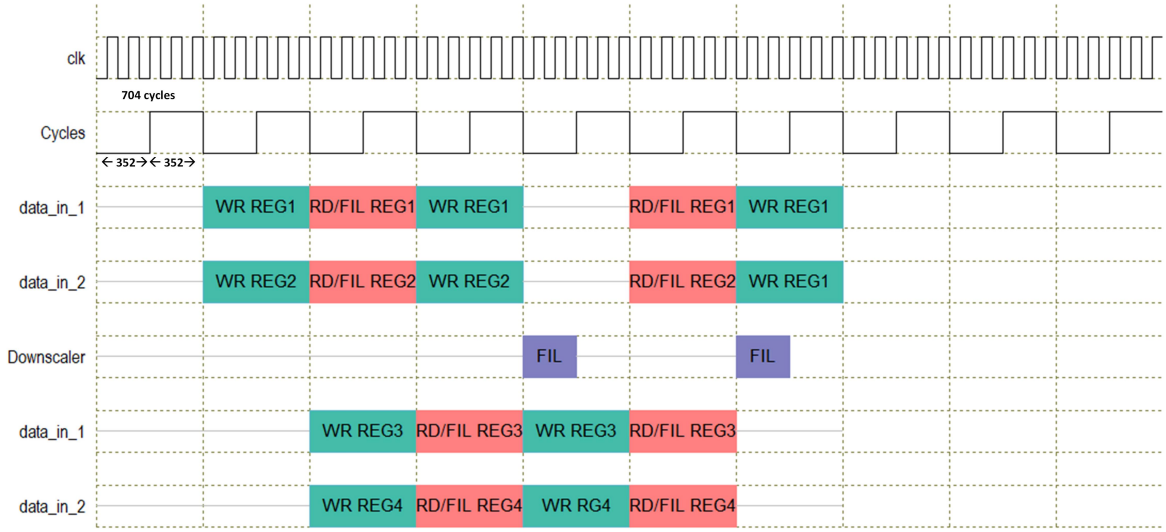

Figure 6.2: Flow of data in the design

These filtered values are store in virtual buffers 1 and 2 as shown in Fig. 6.1. When both the virtual register banks are filled, the final down-scaler is enabled, this filters out the last stage pixels. This is shown in purple color in Fig. 6.2.

Virtual buffer will store exactly half the pixel values compared to the horizontal buffer. Now the filter following the virtual buffer receives input from the previous filter and stores those values as they are computed. Once both the virtual buffer gets filled, the output is generated by further averaging of the virtual pixels.

6.2 Timing diagram

The timing diagram explains the number of clock cycles each block takes for the computation. The Image Down-Scaler module is designed in such a way that it has to wait a certain number of cycle it needs to calculate the end pixel.

6.2.1 Register bank

Fig. 6.3 shows the timing diagram for the horizontal register bank and virtual register bank. Since they are similar in operation this single timing diagram would explain the working of both the register banks.

6.2.2 Filter

Fig.6.4 explains the timing diagram for the filter operation.

6.3 Test-bench Setup

The manipulation of the pixel value is in bits. The section gives detailed information about the image to bits conversion, and vise-versa.

There are separate test-bench's for verification of each lower level module. Apart from the low level testing, there is a test-bench at the top level, which is setup in such a way that it reads the pixel values in rows from a text file generated by a python script used to convert images; details of this script are found in section 6.4. $readmemb is used to read the binary pixel values from the text file. The text file is read at the start of the simulation to initialize the data set register. Then a each clock edge a pixel is fed into the top level module. Apart from the pixel value, the test-bench generates addresses ranging between 0-703. This address is generated in a loop, since they are a series of consecutive numbers.

The test-bench drives the data_in pixel and its corresponding address and the select signal is driven every 704 cycles for the generating internal control signals. The full Image Down-Scaler module is then verified using this top level test bench.

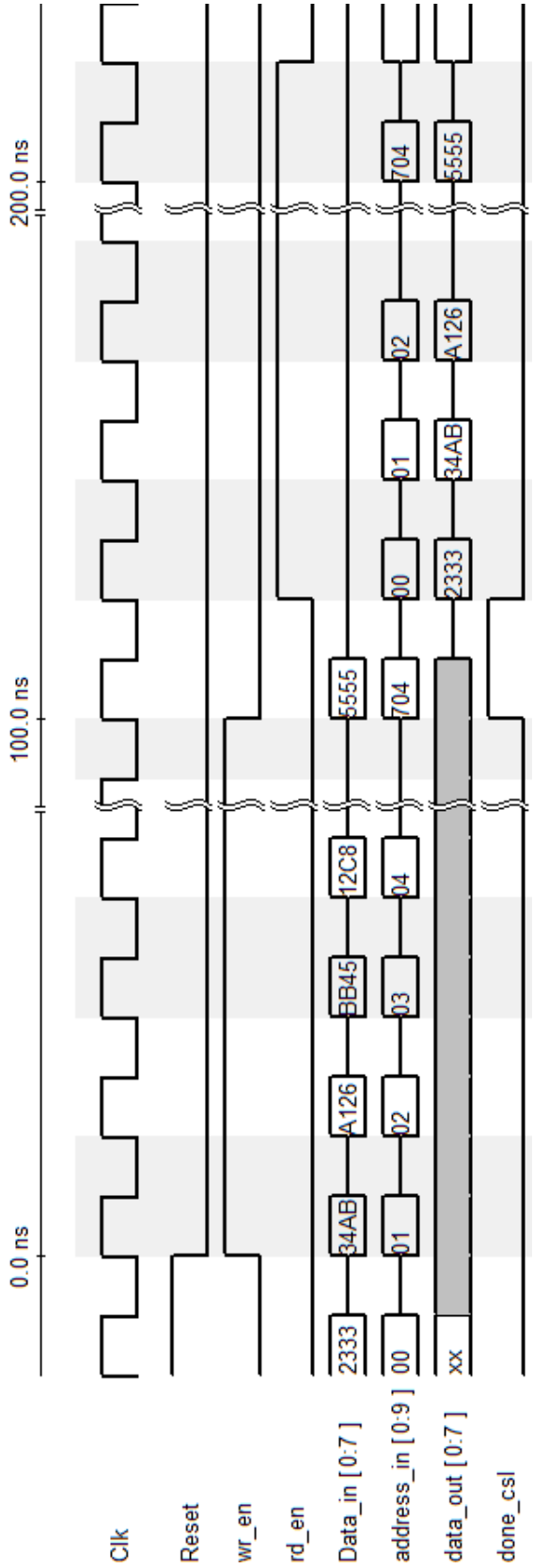

Figure 6.3: Timing diagram for Register bank

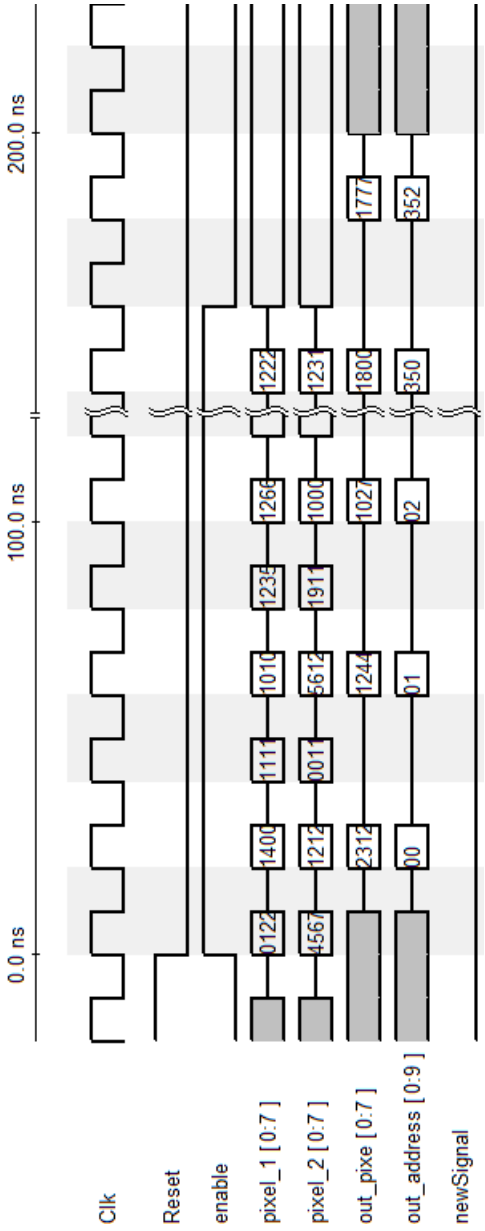

Figure 6.4: Timing diagram for filter

6.4 Python Programs

There are three Python programs used in this project.

- The first one generates the bits from the image. Each pixel value is converted into ASCII bit representation and stored in a text file. One single line represents the value of single pixel. The text file is generated in this way so that it will be easy for the $readmemb to store each pixel value in a data set register.
- The second one is used to convert the bits to an image. The collected output pixel from the test-bench is read by this python script and converts it into image.
- Third one is the implementation of the box filter algorithm as a model for checking operation of the hardware.

All the Python programs are given in Appendix II. Image pixel data are read and written in Python by importing the Python Imaging Library (PIL) package. PIL adds the image processing capability to the Python interpreter [22]. It has a powerful image processing features and it also supports various image formats. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool [22].

Chapter 7

Result and Conclusions

7.1 Image Down-Scaler Module Implementation and Verification

The Image Down-Scaler module is written in the Verilog Hardware Description Language (HDL) designed at the Register Transfer Level (RTL). The lower and higher level modules are synthesized using Synopsis Design Compiler with 32 nm, 65 nm, 90 nm and 180 nm technology node from TMSC. The synthesis step is what transforms the RTL into a gate level netlist, which is a physical description of the hardware consisting of logic gates, standard cells, and their connections [23]. The synthesis is performed by inserting scan chains through out the core, i.e., Design for Testability synthesis. The results of these synthesis are reported in the below section 7.2. The clock frequency used here is $250MHz$.

Before synthesis, the Image Down-Scaler module was simulated using the Cadence Incisive Simulator. The SimVision Debug environment by Cadence helps to analyze the design and test-bench in the verification process. After verifying the Image Down-Scaler module using the test-bench setup mentioned in section 6.3, the design was synthesized

Figure 7.1: Original image

Figure 7.2: Downscaled to 352 X 240 image

using Synopsis synthesis tools. Once synthesized successfully, the resulting netlist was simulated and verified. Though the code is successfully simulated and synthesized, it isn't pipe-lined to output the data every clock cycle. And the implementation can be more effectively optimized by pipe-lining the Image Down-Scaler module completely. And the image that is test does not render the output as expected. But the first level of down scaling works where 704 X 480 is downscaled to 352 X 240 . The Fig. 7.1 shows the original RGB color image, and the Fig. 7.2 shows the single stage downscaled image. But the algorithm implemented in the software produces the image down scaling result as desired.

7.2 Implementation Results

This section gives the results for area, power, and timing for the four different technologies mentioned in the section 7.1. The area distribution report results for the pre-scan netlist are shown in Table 7.2 and for the post-scan netlist in Table 7.3. The pre-scan and post-scan netlist are very similar. The main difference is due to the insertion of scan logic for testing. Inserting scan logic changes the functionality of the design [24], hence there are two different reports. Table 7.4 shows the power report, as technology migrates to smaller geometry leakage contribution to total power consumption increases faster than dynamic power, indicating that leakage will be a major contributor to overall power consumption [25].

For additional details, area of lower level block synthesized in 32 nm library technology are given in the Table 7.1. This shows that the major amount of area is covered by the register banks.

Table 7.1: Area comparison of lower level blocks in 32nm Library $(\mu m)^2$

Technology Node	Down-Scaler	Horizontal register bank	Virtual register bank
Combinational Area	1403	615257	307296
Non-Combinational Area	2601	1182006	591876
Total Area	5004	1797263	899172

Table 7.2: Pre-Scan Netlist synthesis cell area $(\mu m)^2$

Technology Node	im_scaler_32nm	im_scaler_65nm	im_scaler_90nm	im_scaler_180nm
Combinational Area	421722	157357	1908711	3076108
Non-Combinational Area	603247	925288	3203283	5921817
Total Area	1024952	1082646	7108555	8997925

Table 7.3: Post-Scan Netlist synthesis cell area $(\mu m)^2$

Technology Node	im_scaler_32nm	im_scaler_65nm	im_scaler_90nm	im_scaler_180nm
Combinational Area	421722	157146	1908711	3071597
Non-Combinational Area	775604	868562	3203283	7049836
Total Area	2472955	5325709	7108555	8863442

Table 7.4: Pre-scan netlist synthesis Power Report

Technology Node	Internal Power	Switching Power	Dynamic Power	Leakage Power
im_scaler_32nm	28.40mW	405.90μW	28.81mW	117.25mW
im_scaler_65nm	41.80mW	780.09μW	42.55mW	33.626μW
im_scaler_90nm	329.12mW	1.816mW	2.146mW	18.06mW
im_scaler_180nm	280.15mW	13.68mW	293.83mW	35.072μW

Table 7.5: Post-scan netlist synthesis Power Report

Technology Node	Internal Power	Switching Power	Dynamic Power	Leakage Power
im_scaler_32nm	23.40mW	405.90μW	24.29mW	128.22mW
im_scaler_65nm	45.96mW	804.63μW	46.77mW	31.79μW
im_scaler_90nm	4.14mW	3.54mW	7.68mW	22.09mW
im_scaler_180nm	320.70mW	12.84mW	333.55mW	40.33μW

Table 7.6: Pre-scan netlist synthesis Timing Report

Technology Node	Arrival Time (ns)	Slack (ns)
im_scaler_32nm	10.64	8.98
im_scaler_65nm	3.08	16.59
im_scaler_90nm	10.64	8.98
im_scaler_180nm	10.64	8.98

Table 7.7: Post-scan netlist synthesis Timing Report

Technology Node	Arrival Time (ns)	Slack (ns)
im_scaler_32nm	7.20	12.51
im_scaler_65nm	3.08	16.51
im_scaler_90nm	10.60	9.02
im_scaler_180nm	10.62	9.02

7.3 Conclusions

Thus, an Image Down-Scaler module with less computational complexity is implemented in Verilog HDL. This core is synthesized with register banks to hold the pixel values, which reduces the number of memory read and writes if the pixels were accessed from an external memory. There is an increase in the area of this design because of the implementation of the register banks. Note that the Image Down-Scaler module is not completely pipe-lined. The data is output at certain intervals of time. Future work would be to fully pipe-line the Image Down-Scaler module.

References

[1] M. Petrou and C. Petrou, *Image Processing : The Fundamentals*. WILEY, 2010.

[2] C. Dosch, D. Hemingway, and W. Sami, Eds., *Handbook on Digital Terrestrial Television Broadcasti ng Networks and Systems Implementati*. ITU, 2016.

[3] S. Wang and T. Maruyama, "An implementation method of box filter on fpga," *Field Programmable Logic and Applications (FPL), 2016 26th International Conference*, Aug. 2016. [Online]. Available: 10.1109/FPL.2016.7577364

[4] S.M.Mehzabeen and I.Manju, "Efficient optimization of fpga on-chip memory for image processing algorithm," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 2, May 2013.

[5] C. chi Lin, M. hwa Sheu, H. keng Chiang, C. Liaw, and Z. chuan Wu, "The efficient vlsi design of bi-cubic convolution interpolation for digital image processing," *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium*, May 2008. [Online]. Available: 10.1109/ISCAS.2008.4541459

[6] A. Rangsikunpum, E. Leelarasmee, and S. Pumrin, "A design of sign video image expander for hdmi source using bicubic interpolation," *Electrical Engineering/Elec-*

tronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2017 14th International Conference on, Jun. 2017.

[7] C.-H. Kim, S.-M. Seong, J.-A. Lee, and L.-S. Kim, "Winscale: an image-scaling algorithm using an area pixel model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 6, pp. 549–553, June 2003.

[8] M. A. Nuno-Maganda and M. O. Arias-Estrada, "Real-time fpga-based architecture for bicubic interpolation: An application for digital image scaling," *Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference*, Feb. 2005.

[9] P. R. Rajarapollu and V. R. Mankar, "Bicubic interpolation algorithm implementation for image appearance enhancement," *International Journal of Computer Science And Technology*, Jun. 2017. [Online]. Available: http://www.ijcst.com/vol8/8.2/4-prachi-r-rajarapollu.pdf

[10] E. Callway, "Video scaling : Time to banish bilinear!" *Hollywood, CA, USA*, Oct. 2011.

[11] Y. Sa, "Improved bilinear interpolation method for image fast processing," *Intelligent Computation Technology and Automation (ICICTA), 2014 7th International Conference*, Oct. 2014. [Online]. Available: 10.1109/ICICTA.2014.82

[12] M. K. Daniel, V. A. Sophia, and C. J. Moses, "A comparative analysis of bilinear based scalar algorithms," *IEEE*, Mar. 2014.

[13] S. H. Junjie Liu, "L0-regularized image downscaling," *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 27, Mar. 2018. [Online]. Available: 10.1109/TIP.2017.2772838

[14] P. Bourke, "A beginners guide to bitmaps," renderings and models by Peter Diprose and Bill Rattenbury Original November 1993. [Online]. Available: http://paulbourke.net/dataformats/bitmaps/

[15] "A learning community for photographers," Cambridge in color. [Online]. Available: http://www.cambridgeincolour.com/tutorials/bit-depth.htm

[16] L. Walking, "Intoduction to digital resolution," 2011. [Online]. Available: https://www.ala.org.au/wp-content/uploads/2011/10/Introducton-to-Resolution.pdf

[17] E.-L. Tan, W.-S. Gan, and S. Mitra, "Fast arbitrary resizing of images in the discrete cosine transform domain," *IET sponsored by Institution of Engineering and Technology*, Feb. 2011.

[18] L. Alexey, "Tips & tricks: Fast image filtering algorithms," in *GraphiCon'2007*, 2007.

[19] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference*, Dec. 2003.

[20] F. C. Crow, "Summed-area tables for texture mapping," in *Proc. ACM SIGGRAPH*, 1984.

[21] R. Chandel and G. Gupta, "Image filtering algorithms and techniques: A review," *International Journal of Advanced Research in Computer Science and Software Engineering*, Oct. 2013. [Online]. Available: https://pdfs.semanticscholar.org/acc5/3ba7ec21ff7eab60baf9506747c181012d6f.pdf

[22] C. Alex, *Pillow (PIL Fork) Documentation*. Fredrik Lundh and Contributors, 2015. [Online]. Available: https://media.readthedocs.org/pdf/pillow/latest/pillow.pdf

[23] S. L. Harris and D. M. Harris., *Digital Design and Computer Architecture*, A. Edition, Ed. Morgan Kaufmann, 2016.

[24] P. Rashinkar, P. Paterson, and L. Singh, *SYSTEM-ON-A-CHIP VERIFICATION : Methodology and Techniques*, G. Martin and L. Rosenberg, Eds. KLUWER ACADEMIC PUBLISHERS, 2002. [Online]. Available: http://www.ebooks.kluweronline.com/

[25] P. Deepa and C. Vasanthanayaki, "Fpga based efficient on-chip memory for image processing algorithms," *Microelectron.J*, 2012.

Appendix I

Image Down-Scaler Module Verilog Source Code

I.1 Horizontal register bank .v file

```
1
2  module REG_BANK #(parameter width = 23)
3  (
4          input    reset,
5          input    clk,
6          input   [width:0] data_in,
7          input   [9:0] address_in,
8          input   wr_en,
9          input   rd_en,
10         input                test_mode, // DFT test mode control
               signal
11         input                scan_en,   // DFT scan enable signal
12         input                scan_in0,  // DFT scan chain input 0
13         input                scan_in1,  // DFT scan chain input 1
14         input                scan_in2,  // DFT scan chain input 2
15         input                scan_in3,  // DFT scan chain input 3
16         input                scan_in4,  // DFT scan chain input 4
17         output               scan_out0, // DFT scan chain output 0
```

```
18          output                  scan_out1, // DFT scan chain output 1
19          output                  scan_out2, // DFT scan chain output 2
20          output                  scan_out3, // DFT scan chain output 3
21          output                  scan_out4, // DFT scan chain output 4
22          output reg   done_cs,
23          output reg [width:0] data_out
24 );
25 reg [width:0]reg_bank[0:703];
26 always@(posedge clk or posedge reset)
27 begin
28          if(reset)
29                  begin
30                   reg_bank[0] <= 0;reg_bank[1] <= 0;reg_bank[2] <=
                        0;reg_bank[3] <= 0;reg_bank[4] <= 0;reg_bank
                        [5] <= 0;reg_bank[6] <= 0;reg_bank[7] <= 0;
                        reg_bank[8] <= 0;reg_bank[9] <= 0;reg_bank[10]
                         <= 0;reg_bank[11] <= 0;
31                   reg_bank[12] <= 0;reg_bank[13] <= 0;reg_bank[14]
                       <= 0;reg_bank[15] <= 0;reg_bank[16] <= 0;
                        reg_bank[17] <= 0;reg_bank[18] <= 0;reg_bank
                        [19] <= 0;reg_bank[20] <= 0;reg_bank[21] <= 0;
                        reg_bank[22] <= 0;
32                   reg_bank[23] <= 0;reg_bank[24] <= 0;reg_bank[25]
                       <= 0;reg_bank[26] <= 0;reg_bank[27] <= 0;
                        reg_bank[28] <= 0;reg_bank[29] <= 0;reg_bank
                        [30] <= 0;reg_bank[31] <= 0;reg_bank[32] <= 0;
                        reg_bank[33] <= 0;
33                   reg_bank[34] <= 0;reg_bank[35] <= 0;reg_bank[36]
                       <= 0;reg_bank[37] <= 0;reg_bank[38] <= 0;
                        reg_bank[39] <= 0;reg_bank[40] <= 0;reg_bank
                        [41] <= 0;reg_bank[42] <= 0;reg_bank[43] <= 0;
                        reg_bank[44] <= 0;
34                   reg_bank[45] <= 0;reg_bank[46] <= 0;reg_bank[47]
                       <= 0;reg_bank[48] <= 0;reg_bank[49] <= 0;
                        reg_bank[50] <= 0;reg_bank[51] <= 0;reg_bank
                        [52] <= 0;reg_bank[53] <= 0;reg_bank[54] <= 0;
                        reg_bank[55] <= 0;
35                   reg_bank[56] <= 0;reg_bank[57] <= 0;reg_bank[58]
                       <= 0;reg_bank[59] <= 0;reg_bank[60] <= 0;
                        reg_bank[61] <= 0;reg_bank[62] <= 0;reg_bank
                        [63] <= 0;reg_bank[64] <= 0;reg_bank[65] <= 0;
```

```
                reg_bank[66] <= 0;
36              reg_bank[67] <= 0;reg_bank[68] <= 0;reg_bank[69]
                <= 0;reg_bank[70] <= 0;reg_bank[71] <= 0;
                reg_bank[72] <= 0;reg_bank[73] <= 0;reg_bank
                [74] <= 0;reg_bank[75] <= 0;reg_bank[76] <= 0;
                reg_bank[77] <= 0;
37              reg_bank[78] <= 0;reg_bank[79] <= 0;reg_bank[80]
                <= 0;reg_bank[81] <= 0;reg_bank[82] <= 0;
                reg_bank[83] <= 0;reg_bank[84] <= 0;reg_bank
                [85] <= 0;reg_bank[86] <= 0;reg_bank[87] <= 0;
                reg_bank[88] <= 0;
38              reg_bank[89] <= 0;reg_bank[90] <= 0;reg_bank[91]
                <= 0;reg_bank[92] <= 0;reg_bank[93] <= 0;
                reg_bank[94] <= 0;reg_bank[95] <= 0;reg_bank
                [96] <= 0;reg_bank[97] <= 0;reg_bank[98] <= 0;
                reg_bank[99] <= 0;
39              reg_bank[100] <= 0;reg_bank[101] <= 0;reg_bank
                [102] <= 0;reg_bank[103] <= 0;reg_bank[104] <=
                 0;reg_bank[105] <= 0;reg_bank[106] <= 0;
                reg_bank[107] <= 0;reg_bank[108] <= 0;reg_bank
                [109] <= 0;reg_bank[110] <= 0;
40              reg_bank[111] <= 0;reg_bank[112] <= 0;reg_bank
                [113] <= 0;reg_bank[114] <= 0;reg_bank[115] <=
                 0;reg_bank[116] <= 0;reg_bank[117] <= 0;
                reg_bank[118] <= 0;reg_bank[119] <= 0;reg_bank
                [120] <= 0;reg_bank[121] <= 0;
41              reg_bank[122] <= 0;reg_bank[123] <= 0;reg_bank
                [124] <= 0;reg_bank[125] <= 0;reg_bank[126] <=
                 0;reg_bank[127] <= 0;reg_bank[128] <= 0;
                reg_bank[129] <= 0;reg_bank[130] <= 0;reg_bank
                [131] <= 0;reg_bank[132] <= 0;
42              reg_bank[133] <= 0;reg_bank[134] <= 0;reg_bank
                [135] <= 0;reg_bank[136] <= 0;reg_bank[137] <=
                 0;reg_bank[138] <= 0;reg_bank[139] <= 0;
                reg_bank[140] <= 0;reg_bank[141] <= 0;reg_bank
                [142] <= 0;reg_bank[143] <= 0;
43              reg_bank[144] <= 0;reg_bank[145] <= 0;reg_bank
                [146] <= 0;reg_bank[147] <= 0;reg_bank[148] <=
                 0;reg_bank[149] <= 0;reg_bank[150] <= 0;
                reg_bank[151] <= 0;reg_bank[152] <= 0;reg_bank
                [153] <= 0;reg_bank[154] <= 0;
```

```
44                  reg_bank[155] <= 0;reg_bank[156] <= 0;reg_bank
                        [157] <= 0;reg_bank[158] <= 0;reg_bank[159] <=
                         0;reg_bank[160] <= 0;reg_bank[161] <= 0;
                        reg_bank[162] <= 0;reg_bank[163] <= 0;reg_bank
                        [164] <= 0;reg_bank[165] <= 0;
45                  reg_bank[166] <= 0;reg_bank[167] <= 0;reg_bank
                        [168] <= 0;reg_bank[169] <= 0;reg_bank[170] <=
                         0;reg_bank[171] <= 0;reg_bank[172] <= 0;
                        reg_bank[173] <= 0;reg_bank[174] <= 0;reg_bank
                        [175] <= 0;reg_bank[176] <= 0;
46                  reg_bank[177] <= 0;reg_bank[178] <= 0;reg_bank
                        [179] <= 0;reg_bank[180] <= 0;reg_bank[181] <=
                         0;reg_bank[182] <= 0;reg_bank[183] <= 0;
                        reg_bank[184] <= 0;reg_bank[185] <= 0;reg_bank
                        [186] <= 0;reg_bank[187] <= 0;
47                  reg_bank[188] <= 0;reg_bank[189] <= 0;reg_bank
                        [190] <= 0;reg_bank[191] <= 0;reg_bank[192] <=
                         0;reg_bank[193] <= 0;reg_bank[194] <= 0;
                        reg_bank[195] <= 0;reg_bank[196] <= 0;reg_bank
                        [197] <= 0;reg_bank[198] <= 0;
48                  reg_bank[199] <= 0;reg_bank[200] <= 0;reg_bank
                        [201] <= 0;reg_bank[202] <= 0;reg_bank[203] <=
                         0;reg_bank[204] <= 0;reg_bank[205] <= 0;
                        reg_bank[206] <= 0;reg_bank[207] <= 0;reg_bank
                        [208] <= 0;reg_bank[209] <= 0;
49                  reg_bank[210] <= 0;reg_bank[211] <= 0;reg_bank
                        [212] <= 0;reg_bank[213] <= 0;reg_bank[214] <=
                         0;reg_bank[215] <= 0;reg_bank[216] <= 0;
                        reg_bank[217] <= 0;reg_bank[218] <= 0;reg_bank
                        [219] <= 0;reg_bank[220] <= 0;
50                  reg_bank[221] <= 0;reg_bank[222] <= 0;reg_bank
                        [223] <= 0;reg_bank[224] <= 0;reg_bank[225] <=
                         0;reg_bank[226] <= 0;reg_bank[227] <= 0;
                        reg_bank[228] <= 0;reg_bank[229] <= 0;reg_bank
                        [230] <= 0;reg_bank[231] <= 0;
51                  reg_bank[232] <= 0;reg_bank[233] <= 0;reg_bank
                        [234] <= 0;reg_bank[235] <= 0;reg_bank[236] <=
                         0;reg_bank[237] <= 0;reg_bank[238] <= 0;
                        reg_bank[239] <= 0;reg_bank[240] <= 0;reg_bank
                        [241] <= 0;reg_bank[242] <= 0;
```

```
52                  reg_bank[243] <= 0;reg_bank[244] <= 0;reg_bank
                        [245] <= 0;reg_bank[246] <= 0;reg_bank[247] <=
                         0;reg_bank[248] <= 0;reg_bank[249] <= 0;
                        reg_bank[250] <= 0;reg_bank[251] <= 0;reg_bank
                        [252] <= 0;reg_bank[253] <= 0;
53                  reg_bank[254] <= 0;reg_bank[255] <= 0;reg_bank
                        [256] <= 0;reg_bank[257] <= 0;reg_bank[258] <=
                         0;reg_bank[259] <= 0;reg_bank[260] <= 0;
                        reg_bank[261] <= 0;reg_bank[262] <= 0;reg_bank
                        [263] <= 0;reg_bank[264] <= 0;
54                  reg_bank[265] <= 0;reg_bank[266] <= 0;reg_bank
                        [267] <= 0;reg_bank[268] <= 0;reg_bank[269] <=
                         0;reg_bank[270] <= 0;reg_bank[271] <= 0;
                        reg_bank[272] <= 0;reg_bank[273] <= 0;reg_bank
                        [274] <= 0;reg_bank[275] <= 0;
55                  reg_bank[276] <= 0;reg_bank[277] <= 0;reg_bank
                        [278] <= 0;reg_bank[279] <= 0;reg_bank[280] <=
                         0;reg_bank[281] <= 0;reg_bank[282] <= 0;
                        reg_bank[283] <= 0;reg_bank[284] <= 0;reg_bank
                        [285] <= 0;reg_bank[286] <= 0;
56                  reg_bank[287] <= 0;reg_bank[288] <= 0;reg_bank
                        [289] <= 0;reg_bank[290] <= 0;reg_bank[291] <=
                         0;reg_bank[292] <= 0;reg_bank[293] <= 0;
                        reg_bank[294] <= 0;reg_bank[295] <= 0;reg_bank
                        [296] <= 0;reg_bank[297] <= 0;
57                  reg_bank[298] <= 0;reg_bank[299] <= 0;reg_bank
                        [300] <= 0;reg_bank[301] <= 0;reg_bank[302] <=
                         0;reg_bank[303] <= 0;reg_bank[304] <= 0;
                        reg_bank[305] <= 0;reg_bank[306] <= 0;reg_bank
                        [307] <= 0;reg_bank[308] <= 0;
58                  reg_bank[309] <= 0;reg_bank[310] <= 0;reg_bank
                        [311] <= 0;reg_bank[312] <= 0;reg_bank[313] <=
                         0;reg_bank[314] <= 0;reg_bank[315] <= 0;
                        reg_bank[316] <= 0;reg_bank[317] <= 0;reg_bank
                        [318] <= 0;reg_bank[319] <= 0;
59                  reg_bank[320] <= 0;reg_bank[321] <= 0;reg_bank
                        [322] <= 0;reg_bank[323] <= 0;reg_bank[324] <=
                         0;reg_bank[325] <= 0;reg_bank[326] <= 0;
                        reg_bank[327] <= 0;reg_bank[328] <= 0;reg_bank
                        [329] <= 0;reg_bank[330] <= 0;
```

```
60                  reg_bank[331] <= 0;reg_bank[332] <= 0;reg_bank
                        [333] <= 0;reg_bank[334] <= 0;reg_bank[335] <=
                         0;reg_bank[336] <= 0;reg_bank[337] <= 0;
                        reg_bank[338] <= 0;reg_bank[339] <= 0;reg_bank
                        [340] <= 0;reg_bank[341] <= 0;
61                  reg_bank[342] <= 0;reg_bank[343] <= 0;reg_bank
                        [344] <= 0;reg_bank[345] <= 0;reg_bank[346] <=
                         0;reg_bank[347] <= 0;reg_bank[348] <= 0;
                        reg_bank[349] <= 0;reg_bank[350] <= 0;reg_bank
                        [351] <= 0;reg_bank[352] <= 0;
62                  reg_bank[353] <= 0;reg_bank[354] <= 0;reg_bank
                        [355] <= 0;reg_bank[356] <= 0;reg_bank[357] <=
                         0;reg_bank[358] <= 0;reg_bank[359] <= 0;
                        reg_bank[360] <= 0;reg_bank[361] <= 0;reg_bank
                        [362] <= 0;reg_bank[363] <= 0;
63                  reg_bank[364] <= 0;reg_bank[365] <= 0;reg_bank
                        [366] <= 0;reg_bank[367] <= 0;reg_bank[368] <=
                         0;reg_bank[369] <= 0;reg_bank[370] <= 0;
                        reg_bank[371] <= 0;reg_bank[372] <= 0;reg_bank
                        [373] <= 0;reg_bank[374] <= 0;
64                  reg_bank[375] <= 0;reg_bank[376] <= 0;reg_bank
                        [377] <= 0;reg_bank[378] <= 0;reg_bank[379] <=
                         0;reg_bank[380] <= 0;reg_bank[381] <= 0;
                        reg_bank[382] <= 0;reg_bank[383] <= 0;reg_bank
                        [384] <= 0;reg_bank[385] <= 0;
65                  reg_bank[386] <= 0;reg_bank[387] <= 0;reg_bank
                        [388] <= 0;reg_bank[389] <= 0;reg_bank[390] <=
                         0;reg_bank[391] <= 0;reg_bank[392] <= 0;
                        reg_bank[393] <= 0;reg_bank[394] <= 0;reg_bank
                        [395] <= 0;reg_bank[396] <= 0;
66                  reg_bank[397] <= 0;reg_bank[398] <= 0;reg_bank
                        [399] <= 0;reg_bank[400] <= 0;reg_bank[401] <=
                         0;reg_bank[402] <= 0;reg_bank[403] <= 0;
                        reg_bank[404] <= 0;reg_bank[405] <= 0;reg_bank
                        [406] <= 0;reg_bank[407] <= 0;
67                  reg_bank[408] <= 0;reg_bank[409] <= 0;reg_bank
                        [410] <= 0;reg_bank[411] <= 0;reg_bank[412] <=
                         0;reg_bank[413] <= 0;reg_bank[414] <= 0;
                        reg_bank[415] <= 0;reg_bank[416] <= 0;reg_bank
                        [417] <= 0;reg_bank[418] <= 0;
```

```
68                  reg_bank[419] <= 0;reg_bank[420] <= 0;reg_bank
                        [421] <= 0;reg_bank[422] <= 0;reg_bank[423] <=
                         0;reg_bank[424] <= 0;reg_bank[425] <= 0;
                        reg_bank[426] <= 0;reg_bank[427] <= 0;reg_bank
                        [428] <= 0;reg_bank[429] <= 0;
69                  reg_bank[430] <= 0;reg_bank[431] <= 0;reg_bank
                        [432] <= 0;reg_bank[433] <= 0;reg_bank[434] <=
                         0;reg_bank[435] <= 0;reg_bank[436] <= 0;
                        reg_bank[437] <= 0;reg_bank[438] <= 0;reg_bank
                        [439] <= 0;reg_bank[440] <= 0;
70                  reg_bank[441] <= 0;reg_bank[442] <= 0;reg_bank
                        [443] <= 0;reg_bank[444] <= 0;reg_bank[445] <=
                         0;reg_bank[446] <= 0;reg_bank[447] <= 0;
                        reg_bank[448] <= 0;reg_bank[449] <= 0;reg_bank
                        [450] <= 0;reg_bank[451] <= 0;
71                  reg_bank[452] <= 0;reg_bank[453] <= 0;reg_bank
                        [454] <= 0;reg_bank[455] <= 0;reg_bank[456] <=
                         0;reg_bank[457] <= 0;reg_bank[458] <= 0;
                        reg_bank[459] <= 0;reg_bank[460] <= 0;reg_bank
                        [461] <= 0;reg_bank[462] <= 0;
72                  reg_bank[463] <= 0;reg_bank[464] <= 0;reg_bank
                        [465] <= 0;reg_bank[466] <= 0;reg_bank[467] <=
                         0;reg_bank[468] <= 0;reg_bank[469] <= 0;
                        reg_bank[470] <= 0;reg_bank[471] <= 0;reg_bank
                        [472] <= 0;reg_bank[473] <= 0;
73                  reg_bank[474] <= 0;reg_bank[475] <= 0;reg_bank
                        [476] <= 0;reg_bank[477] <= 0;reg_bank[478] <=
                         0;reg_bank[479] <= 0;reg_bank[480] <= 0;
                        reg_bank[481] <= 0;reg_bank[482] <= 0;reg_bank
                        [483] <= 0;reg_bank[484] <= 0;
74                  reg_bank[485] <= 0;reg_bank[486] <= 0;reg_bank
                        [487] <= 0;reg_bank[488] <= 0;reg_bank[489] <=
                         0;reg_bank[490] <= 0;reg_bank[491] <= 0;
                        reg_bank[492] <= 0;reg_bank[493] <= 0;reg_bank
                        [494] <= 0;reg_bank[495] <= 0;
75                  reg_bank[496] <= 0;reg_bank[497] <= 0;reg_bank
                        [498] <= 0;reg_bank[499] <= 0;reg_bank[500] <=
                         0;reg_bank[501] <= 0;reg_bank[502] <= 0;
                        reg_bank[503] <= 0;reg_bank[504] <= 0;reg_bank
                        [505] <= 0;reg_bank[506] <= 0;
```

```
76                  reg_bank[507] <= 0;reg_bank[508] <= 0;reg_bank
                        [509] <= 0;reg_bank[510] <= 0;reg_bank[511] <=
                         0;reg_bank[512] <= 0;reg_bank[513] <= 0;
                        reg_bank[514] <= 0;reg_bank[515] <= 0;reg_bank
                        [516] <= 0;reg_bank[517] <= 0;
77                  reg_bank[518] <= 0;reg_bank[519] <= 0;reg_bank
                        [520] <= 0;reg_bank[521] <= 0;reg_bank[522] <=
                         0;reg_bank[523] <= 0;reg_bank[524] <= 0;
                        reg_bank[525] <= 0;reg_bank[526] <= 0;reg_bank
                        [527] <= 0;reg_bank[528] <= 0;
78                  reg_bank[529] <= 0;reg_bank[530] <= 0;reg_bank
                        [531] <= 0;reg_bank[532] <= 0;reg_bank[533] <=
                         0;reg_bank[534] <= 0;reg_bank[535] <= 0;
                        reg_bank[536] <= 0;reg_bank[537] <= 0;reg_bank
                        [538] <= 0;reg_bank[539] <= 0;
79                  reg_bank[540] <= 0;reg_bank[541] <= 0;reg_bank
                        [542] <= 0;reg_bank[543] <= 0;reg_bank[544] <=
                         0;reg_bank[545] <= 0;reg_bank[546] <= 0;
                        reg_bank[547] <= 0;reg_bank[548] <= 0;reg_bank
                        [549] <= 0;reg_bank[550] <= 0;
80                  reg_bank[551] <= 0;reg_bank[552] <= 0;reg_bank
                        [553] <= 0;reg_bank[554] <= 0;reg_bank[555] <=
                         0;reg_bank[556] <= 0;reg_bank[557] <= 0;
                        reg_bank[558] <= 0;reg_bank[559] <= 0;reg_bank
                        [560] <= 0;reg_bank[561] <= 0;
81                  reg_bank[562] <= 0;reg_bank[563] <= 0;reg_bank
                        [564] <= 0;reg_bank[565] <= 0;reg_bank[566] <=
                         0;reg_bank[567] <= 0;reg_bank[568] <= 0;
                        reg_bank[569] <= 0;reg_bank[570] <= 0;reg_bank
                        [571] <= 0;reg_bank[572] <= 0;
82                  reg_bank[573] <= 0;reg_bank[574] <= 0;reg_bank
                        [575] <= 0;reg_bank[576] <= 0;reg_bank[577] <=
                         0;reg_bank[578] <= 0;reg_bank[579] <= 0;
                        reg_bank[580] <= 0;reg_bank[581] <= 0;reg_bank
                        [582] <= 0;reg_bank[583] <= 0;
83                  reg_bank[584] <= 0;reg_bank[585] <= 0;reg_bank
                        [586] <= 0;reg_bank[587] <= 0;reg_bank[588] <=
                         0;reg_bank[589] <= 0;reg_bank[590] <= 0;
                        reg_bank[591] <= 0;reg_bank[592] <= 0;reg_bank
                        [593] <= 0;reg_bank[594] <= 0;
```

```
84                  reg_bank[595] <= 0;reg_bank[596] <= 0;reg_bank
                        [597] <= 0;reg_bank[598] <= 0;reg_bank[599] <=
                         0;reg_bank[600] <= 0;reg_bank[601] <= 0;
                        reg_bank[602] <= 0;reg_bank[603] <= 0;reg_bank
                        [604] <= 0;reg_bank[605] <= 0;
85                  reg_bank[606] <= 0;reg_bank[607] <= 0;reg_bank
                        [608] <= 0;reg_bank[609] <= 0;reg_bank[610] <=
                         0;reg_bank[611] <= 0;reg_bank[612] <= 0;
                        reg_bank[613] <= 0;reg_bank[614] <= 0;reg_bank
                        [615] <= 0;reg_bank[616] <= 0;
86                  reg_bank[617] <= 0;reg_bank[618] <= 0;reg_bank
                        [619] <= 0;reg_bank[620] <= 0;reg_bank[621] <=
                         0;reg_bank[622] <= 0;reg_bank[623] <= 0;
                        reg_bank[624] <= 0;reg_bank[625] <= 0;reg_bank
                        [626] <= 0;reg_bank[627] <= 0;
87                  reg_bank[628] <= 0;reg_bank[629] <= 0;reg_bank
                        [630] <= 0;reg_bank[631] <= 0;reg_bank[632] <=
                         0;reg_bank[633] <= 0;reg_bank[634] <= 0;
                        reg_bank[635] <= 0;reg_bank[636] <= 0;reg_bank
                        [637] <= 0;reg_bank[638] <= 0;
88                  reg_bank[639] <= 0;reg_bank[640] <= 0;reg_bank
                        [641] <= 0;reg_bank[642] <= 0;reg_bank[643] <=
                         0;reg_bank[644] <= 0;reg_bank[645] <= 0;
                        reg_bank[646] <= 0;reg_bank[647] <= 0;reg_bank
                        [648] <= 0;reg_bank[649] <= 0;
89                  reg_bank[650] <= 0;reg_bank[651] <= 0;reg_bank
                        [652] <= 0;reg_bank[653] <= 0;reg_bank[654] <=
                         0;reg_bank[655] <= 0;reg_bank[656] <= 0;
                        reg_bank[657] <= 0;reg_bank[658] <= 0;reg_bank
                        [659] <= 0;reg_bank[660] <= 0;
90                  reg_bank[661] <= 0;reg_bank[662] <= 0;reg_bank
                        [663] <= 0;reg_bank[664] <= 0;reg_bank[665] <=
                         0;reg_bank[666] <= 0;reg_bank[667] <= 0;
                        reg_bank[668] <= 0;reg_bank[669] <= 0;reg_bank
                        [670] <= 0;reg_bank[671] <= 0;
91                  reg_bank[672] <= 0;reg_bank[673] <= 0;reg_bank
                        [674] <= 0;reg_bank[675] <= 0;reg_bank[676] <=
                         0;reg_bank[677] <= 0;reg_bank[678] <= 0;
                        reg_bank[679] <= 0;reg_bank[680] <= 0;reg_bank
                        [681] <= 0;reg_bank[682] <= 0;
```

```
92                      reg_bank[683] <= 0;reg_bank[684] <= 0;reg_bank
                            [685] <= 0;reg_bank[686] <= 0;reg_bank[687] <=
                             0;reg_bank[688] <= 0;reg_bank[689] <= 0;
                            reg_bank[690] <= 0;reg_bank[691] <= 0;reg_bank
                            [692] <= 0;reg_bank[693] <= 0;
93                      reg_bank[694] <= 0;reg_bank[695] <= 0;reg_bank
                            [696] <= 0;reg_bank[697] <= 0;reg_bank[698] <=
                             0;reg_bank[699] <= 0;reg_bank[700] <= 0;
                            reg_bank[701] <= 0;reg_bank[702] <= 0;reg_bank
                            [703] <= 0;
94                      data_out <= 0;
95            end
96  else
97  begin
98          if (wr_en)
99                  reg_bank[address_in] <= data_in;
100         else if (rd_en)
101
102                 data_out <= reg_bank[address_in];
103 end
104 end
105 always@(posedge clk or posedge reset)
106 begin
107         if(reset)
108                 done_cs <= 0;
109  else
110         begin
111         if(address_in == 10'b1010111111 && wr_en == 1'b1)
112                 done_cs <= 1'b1;
113         else
114                 done_cs <= 0;
115         end
116 end
117 endmodule // REG_BANK
```

I.2 Virtual register bank .v file

```
1
```

```
2  module REG_BANK_2 #(parameter width = 23)
3  (
4          input    reset,
5          input    clk,
6          input   [width:0]data_in,
7          input   [8:0]address_in,
8          input   wr_en,
9          input   rd_en,
10         input                test_mode, // DFT test mode control
               signal
11         input                scan_en,   // DFT scan enable signal
12         input                scan_in0,  // DFT scan chain input 0
13         input                scan_in1,  // DFT scan chain input 1
14         input                scan_in2,  // DFT scan chain input 2
15         input                scan_in3,  // DFT scan chain input 3
16         input                scan_in4,  // DFT scan chain input 4
17         output               scan_out0, // DFT scan chain output 0
18         output               scan_out1, // DFT scan chain output 1
19         output               scan_out2, // DFT scan chain output 2
20         output               scan_out3, // DFT scan chain output 3
21         output               scan_out4, // DFT scan chain output 4
22         output reg   done_cs,
23         output reg [width:0] data_out
24 );
25 reg [width:0]reg_bank[0:351];
26 always@(posedge clk or posedge reset)
27 begin
28         if(reset)
29                 begin
30                  reg_bank[0] <= 0;reg_bank[1] <= 0;reg_bank[2] <=
                        0;reg_bank[3] <= 0;reg_bank[4] <= 0;reg_bank
                        [5] <= 0;reg_bank[6] <= 0;reg_bank[7] <= 0;
                        reg_bank[8] <= 0;reg_bank[9] <= 0;reg_bank[10]
                         <= 0;reg_bank[11] <= 0;
31                  reg_bank[12] <= 0;reg_bank[13] <= 0;reg_bank[14]
                       <= 0;reg_bank[15] <= 0;reg_bank[16] <= 0;
                        reg_bank[17] <= 0;reg_bank[18] <= 0;reg_bank
                        [19] <= 0;reg_bank[20] <= 0;reg_bank[21] <= 0;
                        reg_bank[22] <= 0;
32                  reg_bank[23] <= 0;reg_bank[24] <= 0;reg_bank[25]
                       <= 0;reg_bank[26] <= 0;reg_bank[27] <= 0;
```

```
        reg_bank[28] <= 0;reg_bank[29] <= 0;reg_bank
        [30] <= 0;reg_bank[31] <= 0;reg_bank[32] <= 0;
        reg_bank[33] <= 0;
33      reg_bank[34] <= 0;reg_bank[35] <= 0;reg_bank[36]
        <= 0;reg_bank[37] <= 0;reg_bank[38] <= 0;
        reg_bank[39] <= 0;reg_bank[40] <= 0;reg_bank
        [41] <= 0;reg_bank[42] <= 0;reg_bank[43] <= 0;
        reg_bank[44] <= 0;
34      reg_bank[45] <= 0;reg_bank[46] <= 0;reg_bank[47]
        <= 0;reg_bank[48] <= 0;reg_bank[49] <= 0;
        reg_bank[50] <= 0;reg_bank[51] <= 0;reg_bank
        [52] <= 0;reg_bank[53] <= 0;reg_bank[54] <= 0;
        reg_bank[55] <= 0;
35      reg_bank[56] <= 0;reg_bank[57] <= 0;reg_bank[58]
        <= 0;reg_bank[59] <= 0;reg_bank[60] <= 0;
        reg_bank[61] <= 0;reg_bank[62] <= 0;reg_bank
        [63] <= 0;reg_bank[64] <= 0;reg_bank[65] <= 0;
        reg_bank[66] <= 0;
36      reg_bank[67] <= 0;reg_bank[68] <= 0;reg_bank[69]
        <= 0;reg_bank[70] <= 0;reg_bank[71] <= 0;
        reg_bank[72] <= 0;reg_bank[73] <= 0;reg_bank
        [74] <= 0;reg_bank[75] <= 0;reg_bank[76] <= 0;
        reg_bank[77] <= 0;
37      reg_bank[78] <= 0;reg_bank[79] <= 0;reg_bank[80]
        <= 0;reg_bank[81] <= 0;reg_bank[82] <= 0;
        reg_bank[83] <= 0;reg_bank[84] <= 0;reg_bank
        [85] <= 0;reg_bank[86] <= 0;reg_bank[87] <= 0;
        reg_bank[88] <= 0;
38      reg_bank[89] <= 0;reg_bank[90] <= 0;reg_bank[91]
        <= 0;reg_bank[92] <= 0;reg_bank[93] <= 0;
        reg_bank[94] <= 0;reg_bank[95] <= 0;reg_bank
        [96] <= 0;reg_bank[97] <= 0;reg_bank[98] <= 0;
        reg_bank[99] <= 0;
39      reg_bank[100] <= 0;reg_bank[101] <= 0;reg_bank
        [102] <= 0;reg_bank[103] <= 0;reg_bank[104] <=
         0;reg_bank[105] <= 0;reg_bank[106] <= 0;
        reg_bank[107] <= 0;reg_bank[108] <= 0;reg_bank
        [109] <= 0;reg_bank[110] <= 0;
40      reg_bank[111] <= 0;reg_bank[112] <= 0;reg_bank
        [113] <= 0;reg_bank[114] <= 0;reg_bank[115] <=
         0;reg_bank[116] <= 0;reg_bank[117] <= 0;
```

```
                reg_bank[118] <= 0;reg_bank[119] <= 0;reg_bank
                [120] <= 0;reg_bank[121] <= 0;
41          reg_bank[122] <= 0;reg_bank[123] <= 0;reg_bank
                [124] <= 0;reg_bank[125] <= 0;reg_bank[126] <=
                 0;reg_bank[127] <= 0;reg_bank[128] <= 0;
                reg_bank[129] <= 0;reg_bank[130] <= 0;reg_bank
                [131] <= 0;reg_bank[132] <= 0;
42          reg_bank[133] <= 0;reg_bank[134] <= 0;reg_bank
                [135] <= 0;reg_bank[136] <= 0;reg_bank[137] <=
                 0;reg_bank[138] <= 0;reg_bank[139] <= 0;
                reg_bank[140] <= 0;reg_bank[141] <= 0;reg_bank
                [142] <= 0;reg_bank[143] <= 0;
43          reg_bank[144] <= 0;reg_bank[145] <= 0;reg_bank
                [146] <= 0;reg_bank[147] <= 0;reg_bank[148] <=
                 0;reg_bank[149] <= 0;reg_bank[150] <= 0;
                reg_bank[151] <= 0;reg_bank[152] <= 0;reg_bank
                [153] <= 0;reg_bank[154] <= 0;
44          reg_bank[155] <= 0;reg_bank[156] <= 0;reg_bank
                [157] <= 0;reg_bank[158] <= 0;reg_bank[159] <=
                 0;reg_bank[160] <= 0;reg_bank[161] <= 0;
                reg_bank[162] <= 0;reg_bank[163] <= 0;reg_bank
                [164] <= 0;reg_bank[165] <= 0;
45          reg_bank[166] <= 0;reg_bank[167] <= 0;reg_bank
                [168] <= 0;reg_bank[169] <= 0;reg_bank[170] <=
                 0;reg_bank[171] <= 0;reg_bank[172] <= 0;
                reg_bank[173] <= 0;reg_bank[174] <= 0;reg_bank
                [175] <= 0;reg_bank[176] <= 0;
46          reg_bank[177] <= 0;reg_bank[178] <= 0;reg_bank
                [179] <= 0;reg_bank[180] <= 0;reg_bank[181] <=
                 0;reg_bank[182] <= 0;reg_bank[183] <= 0;
                reg_bank[184] <= 0;reg_bank[185] <= 0;reg_bank
                [186] <= 0;reg_bank[187] <= 0;
47          reg_bank[188] <= 0;reg_bank[189] <= 0;reg_bank
                [190] <= 0;reg_bank[191] <= 0;reg_bank[192] <=
                 0;reg_bank[193] <= 0;reg_bank[194] <= 0;
                reg_bank[195] <= 0;reg_bank[196] <= 0;reg_bank
                [197] <= 0;reg_bank[198] <= 0;
48          reg_bank[199] <= 0;reg_bank[200] <= 0;reg_bank
                [201] <= 0;reg_bank[202] <= 0;reg_bank[203] <=
                 0;reg_bank[204] <= 0;reg_bank[205] <= 0;
                reg_bank[206] <= 0;reg_bank[207] <= 0;reg_bank
```

```
    [208] <= 0;reg_bank[209] <= 0;
49          reg_bank[210] <= 0;reg_bank[211] <= 0;reg_bank
    [212] <= 0;reg_bank[213] <= 0;reg_bank[214] <=
     0;reg_bank[215] <= 0;reg_bank[216] <= 0;
    reg_bank[217] <= 0;reg_bank[218] <= 0;reg_bank
    [219] <= 0;reg_bank[220] <= 0;
50          reg_bank[221] <= 0;reg_bank[222] <= 0;reg_bank
    [223] <= 0;reg_bank[224] <= 0;reg_bank[225] <=
     0;reg_bank[226] <= 0;reg_bank[227] <= 0;
    reg_bank[228] <= 0;reg_bank[229] <= 0;reg_bank
    [230] <= 0;reg_bank[231] <= 0;
51          reg_bank[232] <= 0;reg_bank[233] <= 0;reg_bank
    [234] <= 0;reg_bank[235] <= 0;reg_bank[236] <=
     0;reg_bank[237] <= 0;reg_bank[238] <= 0;
    reg_bank[239] <= 0;reg_bank[240] <= 0;reg_bank
    [241] <= 0;reg_bank[242] <= 0;
52          reg_bank[243] <= 0;reg_bank[244] <= 0;reg_bank
    [245] <= 0;reg_bank[246] <= 0;reg_bank[247] <=
     0;reg_bank[248] <= 0;reg_bank[249] <= 0;
    reg_bank[250] <= 0;reg_bank[251] <= 0;reg_bank
    [252] <= 0;reg_bank[253] <= 0;
53          reg_bank[254] <= 0;reg_bank[255] <= 0;reg_bank
    [256] <= 0;reg_bank[257] <= 0;reg_bank[258] <=
     0;reg_bank[259] <= 0;reg_bank[260] <= 0;
    reg_bank[261] <= 0;reg_bank[262] <= 0;reg_bank
    [263] <= 0;reg_bank[264] <= 0;
54          reg_bank[265] <= 0;reg_bank[266] <= 0;reg_bank
    [267] <= 0;reg_bank[268] <= 0;reg_bank[269] <=
     0;reg_bank[270] <= 0;reg_bank[271] <= 0;
    reg_bank[272] <= 0;reg_bank[273] <= 0;reg_bank
    [274] <= 0;reg_bank[275] <= 0;
55          reg_bank[276] <= 0;reg_bank[277] <= 0;reg_bank
    [278] <= 0;reg_bank[279] <= 0;reg_bank[280] <=
     0;reg_bank[281] <= 0;reg_bank[282] <= 0;
    reg_bank[283] <= 0;reg_bank[284] <= 0;reg_bank
    [285] <= 0;reg_bank[286] <= 0;
56          reg_bank[287] <= 0;reg_bank[288] <= 0;reg_bank
    [289] <= 0;reg_bank[290] <= 0;reg_bank[291] <=
     0;reg_bank[292] <= 0;reg_bank[293] <= 0;
    reg_bank[294] <= 0;reg_bank[295] <= 0;reg_bank
    [296] <= 0;reg_bank[297] <= 0;
```

```
57                    reg_bank[298] <= 0;reg_bank[299] <= 0;reg_bank
                          [300] <= 0;reg_bank[301] <= 0;reg_bank[302] <=
                           0;reg_bank[303] <= 0;reg_bank[304] <= 0;
                          reg_bank[305] <= 0;reg_bank[306] <= 0;reg_bank
                          [307] <= 0;reg_bank[308] <= 0;
58                    reg_bank[309] <= 0;reg_bank[310] <= 0;reg_bank
                          [311] <= 0;reg_bank[312] <= 0;reg_bank[313] <=
                           0;reg_bank[314] <= 0;reg_bank[315] <= 0;
                          reg_bank[316] <= 0;reg_bank[317] <= 0;reg_bank
                          [318] <= 0;reg_bank[319] <= 0;
59                    reg_bank[320] <= 0;reg_bank[321] <= 0;reg_bank
                          [322] <= 0;reg_bank[323] <= 0;reg_bank[324] <=
                           0;reg_bank[325] <= 0;reg_bank[326] <= 0;
                          reg_bank[327] <= 0;reg_bank[328] <= 0;reg_bank
                          [329] <= 0;reg_bank[330] <= 0;
60                    reg_bank[331] <= 0;reg_bank[332] <= 0;reg_bank
                          [333] <= 0;reg_bank[334] <= 0;reg_bank[335] <=
                           0;reg_bank[336] <= 0;reg_bank[337] <= 0;
                          reg_bank[338] <= 0;reg_bank[339] <= 0;reg_bank
                          [340] <= 0;reg_bank[341] <= 0;
61                    reg_bank[342] <= 0;reg_bank[343] <= 0;reg_bank
                          [344] <= 0;reg_bank[345] <= 0;reg_bank[346] <=
                           0;reg_bank[347] <= 0;reg_bank[348] <= 0;
                          reg_bank[349] <= 0;reg_bank[350] <= 0;reg_bank
                          [351] <= 0;
62                    data_out <= 0;
63              end
64    else
65    begin
66          if (wr_en)
67                   reg_bank[address_in] <= data_in;
68          else if (rd_en)
69                   data_out <= reg_bank[address_in];
70    end
71    end
72    always@(posedge clk or posedge reset)
73    begin
74          if(reset)
75                done_cs <= 0;
76     else
77          begin
```

```
78          if(address_in == 9'b101011111 && wr_en == 1'b1)
79                  done_cs <= 1'b1;
80          else
81                  done_cs <= 0;
82          end
83  end
84  endmodule // REG_BANK_2
```

I.3 Filter: Down_scaler .v file

```
1
2   module DOWN_SCALER (
3               input                           reset,
4               input                           clk,
5                   input   [23:0]          pixel_1,
6               input   [23:0]              pixel_2,
7               input                       en,
8                   input                               scan_in0,
9               input                           scan_en,
10              input                           test_mode,
11              output reg                  scan_out0,
12                  output reg [23:0]   out_pixel,
13              output reg [8:0]    out_address,
14              output reg              wr_rd_en
15          );
16  reg count;
17  reg [25:0] num_1, num_2;
18  reg trig_A, trig_B;
19  wire out_en_w;
20  wire [24:0] A;
21  wire count_w;
22  wire [8:0] address_w;
23  reg [8:0] address;
24  reg [8:0] address_B;
25  assign A = ("0" & pixel_1) + ("0" & pixel_2);
26  //*************Pipelining the summed data*************
27  //****************************************************
28  always@(posedge clk or posedge reset)
```

```
29 begin
30         if (reset)
31                 begin
32                         num_1 <= 0;
33                         num_2 <= 0;
34                 end c v
35         else
36                 if(en)
37                         begin
38                                 num_1 <= ( "0" & A );
39                                 num_2 <= num_1;
40                 end
41 end
42 //**************Counter to enable the sum**************
43 //*****************************************************
44 always@(posedge clk or posedge reset)
45 begin
46      if(reset)
47                 count <= 0;
48      else begin
49                         if(count == 0)
50                                 count <= count + 1;
51                         else if (count == 1)
52                                 count <= 0;
53         end
54 end
55 assign count_w = count;
56 //***************Ouput sum and shift ***************
57 //*****************************************************
58 always@(posedge clk or posedge reset)
59 begin
60         if (reset)
61            out_pixel <= 0;
62         else
63         begin
64                  if (count_w)
65                         out_pixel <= out_pixel;
66                  else
67                         out_pixel <= (num_1 + num_2) >> 2;
68         end
69 end
```

```
70  always@(posedge clk or posedge reset)
71  begin
72      if (reset) begin
73          trig_A <= 0;
74          wr_rd_en <= 0;
75      end
76      else begin
77              trig_A <= en;
78              wr_rd_en <= trig_A;
79        end
80  end
81  assign out_en_w = wr_rd_en;
82  always@(posedge clk or posedge reset)
83  begin
84        if(reset)
85                  begin
86                   address <= 0;
87                   address_B <= 0;
88                        out_address <= 0;
89                  end
90        else begin
91                   if(count_w )
92                    begin
93                            address  <= address + 1;
94                    address_B <= address;
95                    end
96                   else if (count_w == 0)
97                   out_address <= address_B;
98                   if (address_w == 9'b101011111)
99                     address <= 0;
100           end
101 end
102 assign address_w = address;
103 endmodule // DOWN_SCALER
```

I.4 im_scaler RGB .v file

```
1
```

```
2  module im_scaler (
3          input    reset,
4          input    clk,
5          input   [23:0]data_in_1,
6          input   [23:0]data_in_2,
7          input   [9:0]address_in,
8          input    sel,
9          input                test_mode, // DFT test mode control
               signal
10         input                scan_en,   // DFT scan enable signal
11         input                scan_in0,  // DFT scan chain input 0
12         input                scan_in1,  // DFT scan chain input 1
13         input                scan_in2,  // DFT scan chain input 2
14         input                scan_in3,  // DFT scan chain input 3
15         input                scan_in4,  // DFT scan chain input 4
16         output               scan_out0, // DFT scan chain output 0
17         output               scan_out1, // DFT scan chain output 1
18         output               scan_out2, // DFT scan chain output 2
19         output               scan_out3, // DFT scan chain output 3
20         output               scan_out4, // DFT scan chain output 4
21         output reg [23:0]    out_pixel
22 );
23 wire u1_done_cs, u2_done_cs, u3_done_cs, u4_done_cs, done_inter_1
       , done_inter_2;
24 wire wr_en_1,wr_en_2,wr_en_3,wr_en_4, wr_en_i_1, wr_en_i_2;
25 wire rd_en_1, rd_en_2, rd_en_3, rd_en_4, rd_en_i_1, rd_en_i_2;
26 wire out_wr_rd_1, out_wr_rd_2, en_1, en_2 , enable;
27 reg u1_wr_en, u2_wr_en, u3_wr_en, u4_wr_en;
28 reg u1_rd_en, u2_rd_en, u3_rd_en, u4_rd_en;
29 wire [23:0] out_pixel_w1;
30 wire [23:0] out_pixel_w2;
31 wire [23:0] out_pixel_w3;
32 wire [23:0] out_pixel_w4;
33 wire [23:0] inter_pixel_1, inter_pixel_2, final_pixel_1,
       final_pixel_2, pixel;
34 wire [8:0] inter_out_addr_1, inter_out_addr_2, out_addr;
35 assign rd_en_1 = u1_rd_en;
36 assign wr_en_1 = u1_wr_en;
37 ////////////////////////////////////////
38 //REG_BANK 1 interface
39 //
```

```
40 REG_BANK u1(
41           .reset             (reset),
42           .clk               (clk),
43           .data_in           (data_in_1),
44           .address_in        (address_in),
45           .wr_en             (wr_en_1),
46           .rd_en             (rd_en_1),
47           .test_mode         (test_mode),
48           .scan_en           (scan_en),
49           .done_cs           (u1_done_cs),
50           .data_out          (out_pixel_w1)
51 );
52 assign rd_en_2 = u2_rd_en;
53 assign wr_en_2 = u2_wr_en;
54 ////////////////////////////////////////
55 //REG_BANK 2 interface
56 //
57 REG_BANK u2(
58           .reset             (reset),
59           .clk               (clk),
60           .data_in           (data_in_1),
61           .address_in        (address_in),
62           .wr_en             (wr_en_2),
63           .rd_en             (rd_en_2),
64           .test_mode         (test_mode),
65           .scan_en           (scan_en),
66           .done_cs           (u2_done_cs),
67           .data_out          (out_pixel_w2)
68 );
69 assign rd_en_3 = u3_rd_en;
70 assign wr_en_3 = u3_wr_en;
71 ////////////////////////////////////////
72 //REG_BANK 3 interface
73 //
74 REG_BANK u3(
75           .reset             (reset),
76           .clk               (clk),
77           .data_in           (data_in_2),
78           .address_in        (address_in),
79           .wr_en             (wr_en_3),
80           .rd_en             (rd_en_3),
```

```
81          .test_mode      (test_mode),
82          .scan_en        (scan_en),
83          .done_cs        (u3_done_cs),
84          .data_out       (out_pixel_w3)
85  );
86  assign rd_en_4 = u4_rd_en;
87  assign wr_en_4 = u4_wr_en;
88  ////////////////////////////////////////////
89  //REG_BANK 4 interface
90  //
91  REG_BANK u4(
92          .reset          (reset),
93          .clk            (clk),
94          .data_in        (data_in_2),
95          .address_in     (address_in),
96          .wr_en          (wr_en_4),
97          .rd_en          (rd_en_4),
98          .test_mode      (test_mode),
99          .scan_en        (scan_en),
100         .done_cs        (u4_done_cs),
101         .data_out       (out_pixel_w4)
102 );
103 assign en_1 = rd_en_1 && rd_en_2;
104 ////////////////////////////////////////////
105 //DOWN_SCALER 1
106 //
107 DOWN_SCALER x1(
108         .reset          (reset),
109         .clk            (clk),
110         .pixel_1        (out_pixel_w1),
111         .pixel_2        (out_pixel_w2),
112         .en             (en_1),
113         .scan_in0       (scan_in0),
114         .scan_en        (scan_en),
115         .test_mode      (test_mode),
116         .out_pixel      (inter_pixel_1),
117         .out_address    (inter_out_addr_1),
118         .wr_rd_en       (out_wr_rd_1)
119     );
120 assign en_2 = rd_en_3 && rd_en_4;
121 ////////////////////////////////////////////
```

```
122 //DOWN_SCALER 2
123 //
124 DOWN_SCALER x2(
125         .reset          (reset),
126         .clk            (clk),
127         .pixel_1        (out_pixel_w3),
128         .pixel_2        (out_pixel_w4),
129         .en             (en_2),
130         .scan_in0       (scan_in0),
131         .scan_en        (scan_en),
132         .test_mode      (test_mode),
133         .out_pixel      (inter_pixel_2),
134         .out_address    (inter_out_addr_2),
135         .wr_rd_en       (out_wr_rd_2)
136     );
137 assign wr_en_i_1 = out_wr_rd_1;
138 assign rd_en_i_1 = ~out_wr_rd_1;
139 ////////////////////////////////////////
140 //REG_BANK 2 interface
141 //
142 REG_BANK_2 w1(
143         .reset          (reset),
144         .clk            (clk),
145         .data_in        (inter_pixel_1),
146         .address_in     (inter_out_addr_1),
147         .wr_en          (wr_en_i_1),
148         .rd_en          (rd_en_i_1),
149         .test_mode      (test_mode),
150         .scan_en        (scan_en),
151         .done_cs        (done_inter_1),
152         .data_out       (final_pixel_1)
153 );
154 assign wr_en_i_2 = out_wr_rd_2;
155 assign rd_en_i_2 = ~out_wr_rd_2;
156 ////////////////////////////////////////
157 //REG_BANK 2 interface
158 //
159 REG_BANK_2 w2(
160         .reset          (reset),
161         .clk            (clk),
162         .data_in        (inter_pixel_2),
```

```
163             .address_in     (inter_out_addr_2),
164             .wr_en          (wr_en_i_2),
165             .rd_en          (rd_en_i_2),
166             .test_mode      (test_mode),
167             .scan_en        (scan_en),
168             .done_cs        (done_inter_2),
169             .data_out       (final_pixel_2)
170 );
171 assign en = 1'b1;
172 ////////////////////////////////////////////
173 //final down scaler which outputs the pixel
174 //
175 DOWN_SCALER f(
176             .reset          (reset),
177             .clk            (clk),
178             .pixel_1        (final_pixel_1),
179             .pixel_2        (final_pixel_2),
180             .en             (en),
181             .scan_in0       (scan_in0),
182             .scan_en        (scan_en),
183             .test_mode      (test_mode),
184             .out_pixel      (pixel),
185             .out_address    (out_addr),
186             .wr_rd_en       (enable)
187         );
188 always@(posedge clk or posedge reset)
189 begin
190             if (reset)
191                     out_pixel <= 0;
192             else
193                    out_pixel <= pixel;
194 end
195 always@(posedge clk or posedge reset)
196 begin
197             if (reset)
198             begin
199                     u1_rd_en <= 0;
200                     u2_rd_en <= 0;
201                     u3_rd_en <= 0;
202                     u4_rd_en <= 0;
203                     u1_wr_en <= 0;
```

```
204                     u2_wr_en <= 0;
205                     u3_wr_en <= 0;
206                     u4_wr_en <= 0;
207             end
208     else
209     begin
210             if (sel)
211             begin
212                     u1_rd_en <=  0;
213                     u2_rd_en <=  0;
214                     u3_rd_en <=  1;
215                     u4_rd_en <=  1;
216                     u1_wr_en <= 1;
217                     u2_wr_en <= 1;
218                     u3_wr_en <= 0;
219                     u4_wr_en <= 0;
220             end
221             else
222             begin
223                     u1_rd_en <=  1;
224                     u2_rd_en <=  1;
225                     u3_rd_en <=  0;
226                     u4_rd_en <=  0;
227                     u1_wr_en <=  0;
228                     u2_wr_en <=  0;
229                     u3_wr_en <=  1;
230                     u4_wr_en <=  1;
231             end
232     end
233     end
234     endmodule // im_scaler
```

I.5 Horizontal register bank test-bench

```
1   module test ();
2           reg clk, reset;
3           reg [7:0] data_in;
4           reg [5:0] address_in;
```

```
5           reg wr_en, rd_en;
6           reg  scan_in0, scan_in1, scan_in2, scan_in3, scan_in4,
               scan_en, test_mode;
7           integer i;
8           wire  scan_out0, scan_out1, scan_out2, scan_out3,
               scan_out4;
9           wire [23:0] data_out;
10          wire done_cs;
11          reg [7:0] even [0:55];
12  REG_BANK top(
13              .reset(reset),
14              .clk(clk),
15              .data_in(data_in),
16              .address_in(address_in),
17              .wr_en(wr_en),
18              .rd_en(rd_en),
19              .test_mode(test_mode),
20              .scan_en(scan_en),
21              .scan_in0(scan_in0),
22              .scan_in1(scan_in1),
23              .scan_in2(scan_in2),
24              .scan_in3(scan_in3),
25              .scan_in4(scan_in4),
26              .scan_out0(scan_out0),
27              .scan_out1(scan_out1),
28              .scan_out2(scan_out2),
29              .scan_out3(scan_out3),
30              .scan_out4(scan_out4),
31              .done_cs(done_cs),
32              .data_out(data_out)
33  );
34  initial
35  begin
36     $timeformat(-9,2,"ns", 16);
37  `ifdef SDFSCAN
38      $sdf_annotate("sdf/REG_BANK_tsmc18_scan.sdf", test.top);
39  `endif
40          clk = 1'b0;
41          reset = 1'b0;
42          wr_en = 1'b0;
43          rd_en = 1'b0;
```

```
44          scan_in0 = 1'b0;
45          scan_in1 = 1'b0;
46          scan_in2 = 1'b0;
47          scan_in3 = 1'b0;
48          scan_in4 = 1'b0;
49          scan_en = 1'b0;
50          test_mode = 1'b0;
51     $readmemb({"../samp.t"}, even);
52 end
53 /* System Clock */
54 always
55 begin
56          #5 clk = ~clk;
57 end
58 initial
59 begin
60       reset = 1'b1;
61       #5;
62       reset = 1'b0;
63       rd_en = 1'b0;
64       wr_en = 1'b1;
65       for ( i = 0; i < 56; i = i + 1)
66       begin
67       #10;
68       data_in = even[i];
69       address_in = i;
70       $display("%d:%h",i,even[i]);
71      end
72      rd_en = 1'b1;
73      wr_en = 1'b0;
74      #20;
75       for ( i = 0; i < 56; i = i + 1)
76       begin
77       #10;
78       data_in = even[i];
79       address_in = i;
80       $display("%d:%h",i,even[i]);
81      end
82       #500;
83       $finish;
84 end
```

```
85
86 endmodule // REG_BANK_test
```

I.6 Virtual register bank test-bench

```
1  module test ();
2          reg clk, reset;
3          reg [7:0] data_in;
4          reg [5:0] address_in;
5          reg wr_en, rd_en;
6          reg  scan_in0, scan_in1, scan_in2, scan_in3, scan_in4,
              scan_en, test_mode;
7          integer i;
8          wire  scan_out0, scan_out1, scan_out2, scan_out3,
              scan_out4;
9          wire [23:0] data_out;
10         wire done_cs;
11         reg [7:0] even [0:55];
12 REG_BANK_2 top(
13             .reset(reset),
14             .clk(clk),
15             .data_in(data_in),
16             .address_in(address_in),
17             .wr_en(wr_en),
18             .rd_en(rd_en),
19             .test_mode(test_mode),
20             .scan_en(scan_en),
21             .scan_in0(scan_in0),
22             .scan_in1(scan_in1),
23             .scan_in2(scan_in2),
24             .scan_in3(scan_in3),
25             .scan_in4(scan_in4),
26             .scan_out0(scan_out0),
27             .scan_out1(scan_out1),
28             .scan_out2(scan_out2),
29             .scan_out3(scan_out3),
30             .scan_out4(scan_out4),
31             .done_cs(done_cs),
```

```
32              .data_out(data_out)
33  );
34  initial
35  begin
36     $timeformat(-9,2,"ns", 16);
37  `ifdef SDFSCAN
38      $sdf_annotate("sdf/REG_BANK_2_tsmc18_scan.sdf", test.top);
39  `endif
40          clk = 1'b0;
41          reset = 1'b0;
42          wr_en = 1'b0;
43          rd_en = 1'b0;
44          scan_in0 = 1'b0;
45          scan_in1 = 1'b0;
46          scan_in2 = 1'b0;
47          scan_in3 = 1'b0;
48          scan_in4 = 1'b0;
49          scan_en = 1'b0;
50          test_mode = 1'b0;
51     $readmemb({"../samp.t"}, even);
52  end
53  /* System Clock */
54  always
55  begin
56          #5 clk = ~clk;
57  end
58  initial
59  begin
60       reset = 1'b1;
61       #5;
62       reset = 1'b0;
63       rd_en = 1'b0;
64       wr_en = 1'b1;
65       for ( i = 0; i < 56; i = i + 1)
66       begin
67       #10;
68       data_in = even[i];
69       address_in = i;
70       $display("%d:%h",i,even[i]);
71      end
72      rd_en = 1'b1;
```

```
73      wr_en = 1'b0;
74      #20;
75       for ( i = 0; i < 56; i = i + 1)
76       begin
77       #10;
78       data_in = even[i];
79       address_in = i;
80       $display("%d:%h",i,even[i]);
81      end
82       #500;
83       $finish;
84 end
85 endmodule // REG_BANK_test
```

I.7 Filter : Dow_scaler test bank test-bench

```
1
2  module test;
3
4  wire  scan_out0;
5  wire [23:0] out_pixel;
6  wire [8:0] out_address;
7  wire wr_rd_en;
8  reg  clk, reset;
9  reg  [23:0] pixel_1, pixel_2;
10 reg  scan_in0, scan_en, test_mode;
11 reg en;
12 DOWN_SCALER top(
13          .reset(reset),
14          .clk(clk),
15          .pixel_1(pixel_1),
16          .pixel_2(pixel_2),
17          .en(en),
18          .scan_in0(scan_in0),
19          .scan_en(scan_en),
20          .test_mode(test_mode),
21          .scan_out0(scan_out0),
22          .out_pixel(out_pixel),
```

```
23          .out_address(out_address),
24          .wr_rd_en(wr_rd_en)
25      );
26  initial
27  begin
28      $timeformat(-9,2,"ns", 16);
29  `ifdef SDFSCAN
30      $sdf_annotate("sdf/DOWN_SCALER_tsmc18_scan.sdf", test.top);
31  `endif
32      clk = 1'b0;
33      reset = 1'b0;
34      scan_in0 = 1'b0;
35      scan_en = 1'b0;
36      test_mode = 1'b0;
37      en = 1'b0;
38  #10 reset = 1'b1;
39  #10 reset = 1'b0;
40  en = 1'b1;
41      repeat (1000)
42          begin
43            pixel_1 = $random;
44            pixel_2 = $random;
45            #10;
46          end
47      $finish;
48  end
49  // 25 MHz clock
50  always
51      #5 clk = ~clk ;
52  endmodule
```

I.8 im_scaler RGB test-bench file

```
1   module test ();
2   // Outputs
3     reg clk, reset;
4     reg [23:0] data_in_1;
5     reg [23:0] data_in_2;
```

```
6    reg[9:0] address_in;
7    reg wr_en, rd_en, sel;
8    integer i, j;
9    wire  scan_out0, scan_out1, scan_out2, scan_out3, scan_out4;
10   wire [23:0] out_pixel;
11   reg  scan_in0, scan_in1, scan_in2, scan_in3, scan_in4, scan_en,
         test_mode;
12   reg [23:0] even [0:168959];
13   reg [23:0] odd  [0:168959];
14 im_scaler top(
15             .reset(reset),
16             .clk(clk),
17             .data_in_1(data_in_1),
18             .data_in_2(data_in_2),
19             .address_in(address_in),
20             .sel(sel),
21             .test_mode(test_mode),
22             .scan_en(scan_en),
23             .scan_in0(scan_in0),
24             .scan_in1(scan_in1),
25             .scan_in2(scan_in2),
26             .scan_in3(scan_in3),
27             .scan_in4(scan_in4),
28             .scan_out0(scan_out0),
29             .scan_out1(scan_out1),
30             .scan_out2(scan_out2),
31             .scan_out3(scan_out3),
32             .scan_out4(scan_out4),
33             .out_pixel(out_pixel)
34 );
35 initial
36 begin
37    $timeformat(-9,2,"ns", 16);
38 `ifdef SDFSCAN
39     $sdf_annotate("sdf/im_scaler_tsmc18_scan.sdf", test.top);
40 `endif
41          clk = 1'b0;
42          reset = 1'b0;
43          scan_in0 = 1'b0;
44          scan_in1 = 1'b0;
45          scan_in2 = 1'b0;
```

```
46          scan_in3 = 1'b0;
47          scan_in4 = 1'b0;
48          scan_en = 1'b0;
49          test_mode = 1'b0;
50     $readmemb({"../pixel_even_row.t"}, even);
51     $readmemb({"../pixel_odd_row.t"}, odd);
52 end
53 /* System Clock */
54 always
55 begin
56 #5 clk = ~clk;
57 end
58          //*****************************************
59          // USAGE:
60          // Input the image text file
61          //
62          //CURRENTLY SETUP TO TEST:
63          // The working of addition of two pixel
64          //*****************************************
65 // initialize the hexadecimal reads from the vectors.txt file
66 /*read and display the values from the text file on screen*/
67 initial begin
68       reset = 1'b1;
69       #50 reset = 1'b0;
70       wr_en = 1'b1;
71       j =0;
72       sel = 1'b1;
73       for ( i = 0; i < 168960; i = i + 1)
74       begin
75       #10;
76       data_in_1 = even[i];
77       data_in_2 = odd[i];
78       $display("%d:%h",i,even[i]);
79       $display("%d:%h",i,odd[i]);
80       address_in = j;
81       if ( j == 703 )
82          begin
83          j = 0;
84          sel = ~sel;
85          end
86       else
```

```
87          j = j+ 1;
88        end
89      #500;
90          $finish;
91    end
92  endmodule // REG_BANK_test
```

I.9 im_scaler Greyscale .v file

```
1
2  module im_scaler_greyscale (
3          input    reset,
4          input    clk,
5          input   [7:0]data_in_1,
6          input   [7:0]data_in_2,
7          input   [8:0]address_in,
8          input    sel,
9          input                test_mode, // DFT test mode control
               signal
10         input                scan_en,   // DFT scan enable signal
11         input                scan_in0,  // DFT scan chain input 0
12         input                scan_in1,  // DFT scan chain input 1
13         input                scan_in2,  // DFT scan chain input 2
14         input                scan_in3,  // DFT scan chain input 3
15         input                scan_in4,  // DFT scan chain input 4
16         output               scan_out0, // DFT scan chain output 0
17         output               scan_out1, // DFT scan chain output 1
18         output               scan_out2, // DFT scan chain output 2
19         output               scan_out3, // DFT scan chain output 3
20         output               scan_out4, // DFT scan chain output 4
21         output reg [8:0]  out_pixel
22 );
23 wire u1_done_cs, u2_done_cs, u3_done_cs, u4_done_cs, done_inter_1
       , done_inter_2;
24 wire wr_en_1,wr_en_2,wr_en_3,wr_en_4, wr_en_i_1, wr_en_i_2;
25 wire rd_en_1, rd_en_2, rd_en_3, rd_en_4, rd_en_i_1, rd_en_i_2;
26 wire out_wr_rd_1, out_wr_rd_2, en_1, en_2 , enable;
27 reg u1_wr_en, u2_wr_en, u3_wr_en, u4_wr_en;
```

```
28 reg u1_rd_en, u2_rd_en, u3_rd_en, u4_rd_en;
29 wire [8:0] out_pixel_w1;
30 wire [8:0] out_pixel_w2;
31 wire [8:0] out_pixel_w3;
32 wire [8:0] out_pixel_w4;
33 wire [8:0] inter_pixel_1, inter_pixel_2, final_pixel_1,
      final_pixel_2, pixel;
34 wire [8:0] inter_out_addr_1, inter_out_addr_2, out_addr;
35 assign rd_en_1 = u1_rd_en;
36 assign wr_en_1 = u1_wr_en;
37 ////////////////////////////////////////
38 //REG_BANK 1 interface
39 //
40 REG_BANK u1(
41          .reset          (reset),
42          .clk            (clk),
43          .data_in        (data_in_1),
44          .address_in     (address_in),
45          .wr_en          (wr_en_1),
46          .rd_en          (rd_en_1),
47          .test_mode      (test_mode),
48          .scan_en        (scan_en),
49          .done_cs        (u1_done_cs),
50          .data_out       (out_pixel_w1)
51 );
52 assign rd_en_2 = u2_rd_en;
53 assign wr_en_2 = u2_wr_en;
54 ////////////////////////////////////////
55 //REG_BANK 2 interface
56 //
57 REG_BANK u2(
58          .reset          (reset),
59          .clk            (clk),
60          .data_in        (data_in_1),
61          .address_in     (address_in),
62          .wr_en          (wr_en_2),
63          .rd_en          (rd_en_2),
64          .test_mode      (test_mode),
65          .scan_en        (scan_en),
66          .done_cs        (u2_done_cs),
67          .data_out       (out_pixel_w2)
```

```
68  );
69  assign rd_en_3 = u3_rd_en;
70  assign wr_en_3 = u3_wr_en;
71  ////////////////////////////////////////////
72  //REG_BANK 3 interface
73  //
74  REG_BANK u3(
75          .reset          (reset),
76          .clk            (clk),
77          .data_in        (data_in_2),
78          .address_in     (address_in),
79          .wr_en          (wr_en_3),
80          .rd_en          (rd_en_3),
81          .test_mode      (test_mode),
82          .scan_en        (scan_en),
83          .done_cs        (u3_done_cs),
84          .data_out       (out_pixel_w3)
85  );
86  assign rd_en_4 = u4_rd_en;
87  assign wr_en_4 = u4_wr_en;
88  ////////////////////////////////////////////
89  //REG_BANK 4 interface
90  //
91  REG_BANK u4(
92          .reset          (reset),
93          .clk            (clk),
94          .data_in        (data_in_2),
95          .address_in     (address_in),
96          .wr_en          (wr_en_4),
97          .rd_en          (rd_en_4),
98          .test_mode      (test_mode),
99          .scan_en        (scan_en),
100         .done_cs        (u4_done_cs),
101         .data_out       (out_pixel_w4)
102 );
103 assign en_1 = rd_en_1 && rd_en_2;
104 ////////////////////////////////////////////
105 //DOWN_SCALER 1
106 //
107 DOWN_SCALER x1(
108         .reset          (reset),
```

```
109             .clk                (clk),
110             .pixel_1            (out_pixel_w1),
111             .pixel_2            (out_pixel_w2),
112             .en                 (en_1),
113             .scan_in0           (scan_in0),
114             .scan_en            (scan_en),
115             .test_mode          (test_mode),
116             .out_pixel          (inter_pixel_1),
117             .out_address        (inter_out_addr_1),
118             .wr_rd_en           (out_wr_rd_1)
119         );
120 assign en_2 = rd_en_3 && rd_en_4;
121 ////////////////////////////////////////////
122 //DOWN_SCALER 2
123 //
124 DOWN_SCALER x2(
125             .reset              (reset),
126             .clk                (clk),
127             .pixel_1            (out_pixel_w3),
128             .pixel_2            (out_pixel_w4),
129             .en                 (en_2),
130             .scan_in0           (scan_in0),
131             .scan_en            (scan_en),
132             .test_mode          (test_mode),
133             .out_pixel          (inter_pixel_2),
134             .out_address        (inter_out_addr_2),
135             .wr_rd_en           (out_wr_rd_2)
136         );
137 assign wr_en_i_1 = out_wr_rd_1;
138 assign rd_en_i_1 = ~out_wr_rd_1;
139 ////////////////////////////////////////////
140 //REG_BANK 2 interface
141 //
142 REG_BANK_2 w1(
143             .reset              (reset),
144             .clk                (clk),
145             .data_in            (inter_pixel_1),
146             .address_in         (inter_out_addr_1),
147             .wr_en              (wr_en_i_1),
148             .rd_en              (rd_en_i_1),
149             .test_mode          (test_mode),
```

```
150             .scan_en            (scan_en),
151             .done_cs            (done_inter_1),
152             .data_out           (final_pixel_1)
153     );
154     assign wr_en_i_2 = out_wr_rd_2;
155     assign rd_en_i_2 = ~out_wr_rd_2;
156     ////////////////////////////////////////////
157     //REG_BANK 2 interface
158     //
159     REG_BANK_2 w2(
160             .reset              (reset),
161             .clk                (clk),
162             .data_in            (inter_pixel_2),
163             .address_in         (inter_out_addr_2),
164             .wr_en              (wr_en_i_2),
165             .rd_en              (rd_en_i_2),
166             .test_mode          (test_mode),
167             .scan_en            (scan_en),
168             .done_cs            (done_inter_2),
169             .data_out           (final_pixel_2)
170     );
171     assign en = 1'b1;
172     ////////////////////////////////////////////
173     //final down scaler which outputs the pixel
174     //
175     DOWN_SCALER f(
176             .reset              (reset),
177             .clk                (clk),
178             .pixel_1            (final_pixel_1),
179             .pixel_2            (final_pixel_2),
180             .en                 (en),
181             .scan_in0           (scan_in0),
182             .scan_en            (scan_en),
183             .test_mode          (test_mode),
184             .out_pixel          (pixel),
185             .out_address        (out_addr),
186             .wr_rd_en           (enable)
187         );
188     always@(posedge clk or posedge reset)
189     begin
190             if (reset)
```

```
191                     out_pixel <= 0;
192             else
193                     out_pixel <= pixel;
194 end
195 always@(posedge clk or posedge reset)
196 begin
197             if (reset)
198             begin
199                     u1_rd_en <= 0;
200                     u2_rd_en <= 0;
201                     u3_rd_en <= 0;
202                     u4_rd_en <= 0;
203                     u1_wr_en <= 0;
204                     u2_wr_en <= 0;
205                     u3_wr_en <= 0;
206                     u4_wr_en <= 0;
207             end
208 else
209 begin
210             if (sel)
211             begin
212                     u1_rd_en <=  0;
213                     u2_rd_en <=  0;
214                     u3_rd_en <=  1;
215                     u4_rd_en <=  1;
216                     u1_wr_en <= 1;
217                     u2_wr_en <= 1;
218                     u3_wr_en <= 0;
219                     u4_wr_en <= 0;
220             end
221             else
222
223                     u1_rd_en <=  1;
224                     u2_rd_en <=  1;
225                     u3_rd_en <=  0;
226                     u4_rd_en <=  0;
227                     u1_wr_en <=  0;
228                     u2_wr_en <=  0;
229                     u3_wr_en <=  1;
230                     u4_wr_en <=  1;
231             end
```

```
232 end
233 end
234 endmodule // im_scaler_greyscale
```

I.10 im_scaler Greyscale test-bench

```
1  module test ();
2  // Outputs
3    reg clk, reset;
4    reg [7:0] data_in_1;
5    reg [7:0] data_in_2;
6    reg[8:0] address_in;
7    reg wr_en, rd_en, sel;
8    integer i, j;
9    wire  scan_out0, scan_out1, scan_out2, scan_out3, scan_out4;
10   wire [8:0] out_pixel;
11   reg  scan_in0, scan_in1, scan_in2, scan_in3, scan_in4, scan_en,
         test_mode;
12   reg [23:0] even [0:168959];
13   reg [23:0] odd  [0:168959];
14 im_scaler_greyscale top(
15             .reset(reset),
16             .clk(clk),
17             .data_in_1(data_in_1),
18             .data_in_2(data_in_2),
19             .address_in(address_in),
20             .sel(sel),
21             .test_mode(test_mode),
22             .scan_en(scan_en),
23             .scan_in0(scan_in0),
24             .scan_in1(scan_in1),
25             .scan_in2(scan_in2),
26             .scan_in3(scan_in3),
27             .scan_in4(scan_in4),
28             .scan_out0(scan_out0),
29             .scan_out1(scan_out1),
30             .scan_out2(scan_out2),
31             .scan_out3(scan_out3),
```

```
32              .scan_out4(scan_out4),
33              .out_pixel(out_pixel)
34  );
35  initial
36  begin
37     $timeformat(-9,2,"ns", 16);
38  `ifdef SDFSCAN
39      $sdf_annotate("sdf/im_scaler_greyscale_tsmc18_scan.sdf", test
            .top);
40  `endif
41           clk = 1'b0;
42           reset = 1'b0;
43           scan_in0 = 1'b0;
44           scan_in1 = 1'b0;
45           scan_in2 = 1'b0;
46           scan_in3 = 1'b0;
47           scan_in4 = 1'b0;
48           scan_en = 1'b0;
49           test_mode = 1'b0;
50     $readmemb({"../pixel_even_row.t"}, even);
51     $readmemb({"../pixel_odd_row.t"}, odd);
52  end
53  /* System Clock */
54  always
55  begin
56  #5 clk = ~clk;
57  end
58           //*****************************************
59           // USAGE:
60           // Input the image text file
61           //
62           //CURRENTLY SETUP TO TEST:
63           // The working of addition of two pixel
64           //*****************************************
65  // initialize the hexadecimal reads from the vectors.txt file
66  /*read and display the values from the text file on screen*/
67  initial begin
68        reset = 1'b1;
69        #50 reset = 1'b0;
70        wr_en = 1'b1;
71        j =0;
```

```
72          sel = 1'b1;
73          for ( i = 0; i < 168960; i = i + 1)
74          begin
75          #10;
76          data_in_1 = even[i];
77          data_in_2 = odd[i];
78          $display("%d:%h",i,even[i]);
79          $display("%d:%h",i,odd[i]);
80          address_in = j;
81          if ( j == 703 )
82              begin
83              j = 0;
84              sel = ~sel;
85              end
86          else
87              j = j+ 1;
88          end
89       #500;
90              $finish;
91      end
92  endmodule // im_scaler_greyscale_test
```

Appendix II

Python Script Files

II.1 Image to bits conversion file

```
if __name__ == '__main__':
    from PIL import Image
    data = []
    out = []
    out1 = []
    out_next = []
    out_next_1 = []
    im = Image.open("home.jpg")
    col, row = im.size
    pixels = im.load()
    f1 = open("pixel_even_row.t", "w")
    f2 = open("pixel_odd_row.t", "w")
    for i in range(0, 480):
        for j in range(0, 704):
            r, b, g = pixels[j, i]
            R = '{0:08b}'.format(r)
            G = '{0:08b}'.format(g)
            B = '{0:08b}'.format(b)
            if (i == 0):
                f1.write(R)
                f1.write(G)
```

```
                f1.write(B)
                f1.write("\n")
            elif ((i % 2) == 0):
                f1.write(R)
                f1.write(G)
                f1.write(B)
                f1.write("\n")
            else:
                f2.write(R)
                f2.write(G)
                f2.write(B)
                f2.write("\n")
```

II.2 Python implementation of the Filter

```
##########################################################
# IMAGE DOWNSCALING BY AVERAGING PIXELS
# Description: converts 704x480 pixels to
# 352x240 pixels by averaging and then does
# conversion to 176x120 pixels
#
#Autor: vaishnavi parthipan
##########################################################
if __name__ == '__main__':
    from PIL import Image
    data = []
    out = []
    out1 = []
    out_next = []
    out_next_1 = []
    im = Image.open("new.jpg")
    col, row = im.size
    pixels = im.load()
    for i in range(0, 479, 2):
        for j in range(0, 703, 2):
            r, b, g = pixels[j, i]
            r1, g1, b1 = pixels[j+1, i]
            r2, g2, b2 = pixels[j, i+1]
```

```
24                r3, g3, b3 = pixels[j+1, i+1]
25                R = (r+r1+r2+r3)//4
26                G = (g+g1+g2+g3)//4
27                B = (b+b1+b2+b3)//4
28                data.append(R)
29                data.append(G)
30                data.append(B)
31        out = tuple(data)
32        for i in range(0, len(out), 3):
33            out_next.append(out[i:i + 3])
34        print(out_next)
35        NEW_X_SIZE = 352
36        NEW_Y_SIZE = 240
37        image_out = Image.new('RGB', (NEW_X_SIZE, NEW_Y_SIZE))
38        image_out.putdata(out_next)
39        image_out.save('352x240.png')
40    ##########################################################################
41        data1 =[]
42        im = Image.open("352x240.png")
43        col, row = im.size
44        pixels = im.load()
45        for i in range(0, 239, 2):
46            for j in range(0, 351, 2):
47                r, b, g = pixels[j, i]
48                r1, g1, b1 = pixels[j + 1, i]
49                r2, g2, b2 = pixels[j, i + 1]
50                r3, g3, b3 = pixels[j + 1, i + 1]
51                R = (r + r1 + r2 + r3) // 4
52                G = (g + g1 + g2 + g3) // 4
53                B = (b + b1 + b2 + b3) // 4
54                data1.append(R)
55                data1.append(G)
56                data1.append(B)
57        out1 = tuple(data1)
58        for i in range(0, len(out1), 3):
59            out_next_1.append(out1[i:i + 3])
60        # print(out_next)
61        NEW_X_SIZE = 176
62        NEW_Y_SIZE = 120
63        image_out = Image.new('RGB', (NEW_X_SIZE, NEW_Y_SIZE))
64        image_out.putdata(out_next_1)
```

```
65      image_out.save('176x120.png')
```

II.3 Bits to image conversion python

```
1  if __name__ == '__main__':
2      from PIL import Image
3      filename = 'output.txt'
4      with open(filename) as f:
5          data[x] = f.read()
6      img = Image.new('1', (176, 120))
7      im = Image.fromarray(data, 'RGB')
8      img.putdata(data)
9      im.show()
```